خطوة على طريق

فيجوال بيزك دوت نت

سامح السيد كامل

خطوة على طريق فيجوال بيزك دوت نت

لمن هذا الكتاب:

هذا الكتاب موجه لأحد الفئات التالية

- من يريد تعلم البرمجة وليس لديه سابق معرفه بها.
- المُعلم الذى يجد صعوبة فى تدريس مناهج البرمجة الخاصة بفيجوال بيزك دوت نت.
- الطالب الذى يريد تعلم البرمجة بطريقة مختلفة عن طريقة عرض الكتاب المدرسى.

ملخص ما يقدمه الكتاب:

يأخذك الكتاب فى رحلة قصيرة تمر فيها على خمس محطات، المحطة الأولى تزودك بمعلومات عن ماهية البرمجة ومفاهيمها، مراحل تطوير النظم وخطوات سير البرامج، التعرف على منصة التطوير (دوت نت) وما بها من لغات برمجية وأدوات تطوير، ثم ينتهى بك إلى مراحل ترجمة التطبيق ليظهر أمامك فتراه بعينيك وتلمسه بمؤشر فأرتك أو بإصبعك إن كنت تتعامل مع Touch Screen.

ثم يصل بك إلى المحطة الثانية والتى تُعد حجر الأساس الذى يمثل أهم عنصر فى جميع العمليات الإلكترونية، وهى البيانات.

ثم تنتقل بعدها إلى المحطة الثالثة والتى تتعرف فيها على بناء الواجهة الرسومية للتطبيق باستخدام فئات Windows Forms.

حتى تصل إلى المحطة الرابعة وهى محطة جوهرية فى لغات البرمجة والتى تستطيع من خلالها التحكم فى مسار البرنامج وتنفيذ الأوامر ومعالجة الأخطاء الغير متوقعة، وذلك عن طريق مجموعة من التراكيب البرمجية تسمى هياكل القرارات وحلقات التكرار.

إلى هنا تصل إلى المحطة الخامسة والتى تمثل التطبيق العملى لكل ما ورد فى المحطات السابقة، حيث ستبدأ فى التعرف على النواة الأساسية لبناء التطبيقات عن طريق كتابة الإجراءات والدوال والتى تعتبر اللبنات التى تبنى بها البرامج.

فى النهاية عليك أن تعلم أن الكتاب لن يقدم لك كل شىء ولا أى كتاب آخر ، فمهمة الكتاب تقتصر على أن تضع قدميك على أول الطريق وبعدها عليك أن تكمل مشوارك معتمداً على نفسك ، مع مزيد من الصبر والجهد والعزيمة ستصل إلى ما تريد ، المهم أن تحدد هدفك وتسعى جاهداً لتحقيقه ، ولا تنسى أنك ستواجه صعوبات كثيرة وعناء وستقابلك حواجز ، فى البداية ستشعر وكأنك تسير على قدميك بترقب وحذر لذا ستكون حركتك بطيئة ، مع الوقت تزداد معلوماتك وتُكثر من الممارسة والتطبيق ستجد نفسك أسرع وأكثر ثقة وكأنك تركب فرساً أبجر كعنترة بن شداد.

الأدوات المطلوبة لمتابعة وتطبيق الأمثلة العملية:

- بيئة التطوير Microsoft Visual Studio أى إصدار ، ويمكنك استخدام النسخ المجانية Express Editions ويمكنك تحميلها من موقع visualstudio.com وستجد أحدث إصدار مجانى حالياً هو Visual Studio Community 2015 RC

المقدمة

تعلُم البرمجة هو شيء مفيد فى حد ذاته ، فالبرمجة لن تفيدك فى عملك فحسب بل ستوسع من إدراكك ببواطن الأمور والتى تجرى فى الخفاء خلف مجموعة من الصور والأشكال الأنيقة التى تظهر على شاشة الحاسب وأيضا ستجعلك أكثر قدرة على إنجاز العديد من المهام إستناداً إلى الفكر البرمجى الذى سينمو بداخلك من تعلم البرمجة ، وهذا الهدف هو نفسه المقصود بالنسبة للطفل فى مراحلة الأولى من التعليم لتهيئته منذ الصغر وتأسيسه على أساس علمى متين يستطيع من خلاله الدخول إلى عالم المستقبل ، ولكن الأمر يبدو مختلفاً بالنسبة للمعلم ، فالطالب يقع فى جانب المُتلقى للمعلومات أما المعلم فهو المرشد والموجه والناصح والمُوضِح ، لذا يجب أن يكون مؤهلاً بالقدر الكافى من المعلومات والمهارات والممارسات التى تمكنه من إنجاز مهمته على أكمل وجه ، والعبء الأكبر فى بناء المخزون المعرفى والثقافى والمهارى للمعلم يقع عليه ذاته ، فمهما دُعِم المعلم بالدورات التدريبية والكتب والدروس التعليمية لن يفيد إذا لم يكن لديه الاستعداد والرغبة الداخلية للتعلم ، ولا يجب أن ينتهى الأمر عند هذا فحسب ولكن يجب على المعلم أن يكمل بنائه المعرفى بالبحث والتمحيص والتمرين فإذا وجدت من يأخذ بيدك لتصعد إلى أول درجة من سُلم المعرفة فيجب عليك أن تعتمد على نفسك وتصعد إلى الطابق الأول ثم الثانى ولا تمل ولا تكسل فالبناء عالى جداً ويعلو يوماً بعد يوم وربما لا ينتهى إلا بنهاية العالم.

أحياناً يتفوق التلميذ على أستاذه ، فكثير من الاختراعات والابتكارات سواء كانت فى علوم الحاسب وتكنولوجيا المعلومات أو فى أى مجال آخر ، ولدت على يد علماء وباحثين وأساتذة جامعات ، لكن هناك اختراعات لأطفال وشباب صغار وأشخاص عاديين غيرت حياة الإنسان ، وما كانت فى بدايتها إلا فكرة ومضت فى الخيال ، أو حُلم فى المنام ، أو ربما كانت حُلم فى اليقظة.

سامح كامل

٢٠١٤/٢/٣٠م



المحطة الأولى

مقدمة عن البرمجة

هذه الآلة التى لا تفهم ولا تعى شيئاً والتى أطلقوا عليها اسم الحاسب (Computer) والتى تتكون من مجموعة من الأجزاء المرئية والملموسة مثل الشاشة ولوحة المفاتيح والفأرة وهذه العُلبة التى تحوى بداخلها الكثير من التفاصيل وبها الذاكرة والمعالج ومحركات الأقراص واللوحة الأم والتى تظهر عليها العديد من الدوائر الإلكترونية والمسارات البارزة والأسلاك والكابلات ، كل هذا يخفى بداخله أشياء بالغة التعقيد ، فلو حركت مؤشر الفأرة على سطح المكتب وأشرت إلى أحد الرموز فيظهر تلميح باسم هذا الرمز أو وظيفته ، ولو فتحت لعبة معينة كلعبة السيارات واستخدمت الأسهم الموجودة بلوحة المفاتيح لتقود السيارة وتحركها يميناً ويساراً لكان أمراً مسلياً وممتعاً بالفعل ، فهل لك أن تخبرنى بتفاصيل ما حدث فى الخلفية من أوامر وتعليمات جعلت الآلة تفهم وتنفذ تلك الأوامر ، هذا هو ما لا يمكن أن تراه وتلمسه بيدك إنها البرمجة (Programming).

مفاهيم البرمجة

البرمجة Programming

البرمجة هى القدرة على مخاطبة الحاسب لينفذ مهاماً جديدة لا يمكن له أن ينفذها من تلقاء ذاته ، فهى تمنحك القدرة على التحكم بجميع الأجهزة المرتبطة به والسيطرة عليها ، وبمفهوم أبسط فالبرمجة هى شرح تفصيلى لما يجب أن يقوم به الحاسب من مهام ، والإنسان هو من يقوم بهذه المهمة وهذا هو ما يطلق عليه المبرمج (Programmer) ، وهل يستطيع الحاسب فهم لغة البشر؟! بالتأكيد لا ، فالحاسب لا يفهم سوى لغة واحدة وهى لغة الكهرباء والمتمثلة فى إشارتين (أقصى وأدنى جهد محتمل) يفهم الحاسب هذه الإشارات بالمنطق أى إما True أو False وتمثل بالأرقام (١ و ٠) وتمثل رياضياً بما يسمى النظام الثنائى وهذا ما يطلق عليه لغة الآلة (Machine Language) ، نستنتج من ذلك أن جميع البيانات المدخلة إلى الحاسب وجميع الأوامر الموجهه إليه يجب أن تحول من صورتها الأصلية إلى سلسلة من الأرقام بالنظام الثنائى حتى يفهمها الحاسب وينفذها.

الحاسب الذى نتحدث عنه ليس فقط جهاز الكمبيوتر ، وإنما هو أى جهاز إلكترونى سواء كان كمبيوتر أو موبايل أو تلفزيون أو الآلة الحاسبة التقليدية أو حتى ساعة اليد الرقمية.



لغات البرمجة Programming Language

لغات البرمجة كلغات البشر فهى وسيلة للتخاطب والتعارف ، عندما بدأ البشر فى التعبير تحدثوا ببعض الأصوات تعارفوا عليها فيما بينهم ومع تطور الإنسانية قاموا بتمثيل هذه الأصوات برموز مكتوبة إلى أن وصلنا إلى المئات من لغات البشر التى لكل منها رموز خاصة وقواعد كتابة ونطق ، وإذا أراد شخص يتحدث اللغة العربية أن يخاطب شخصاً يتحدث الإنجليزية فيجب علي أحدهما أن يتعلم لغة الآخر أو يستعين بشخص ثالث يتحدث اللغتين حتى يعمل كوسيط بينهما ، الآن يمكنك أن تستوعب مفهوم لغة البرمجة بكل سهولة فهى عبارة عن مجموعة من الكلمات والرموز التى تكتب وفق قواعد محددة لتُكوِن عبارات وجمل لها معنى محدد يتم تمريرها إلى ما يسمى بالمترجم (Compiler) ليقوم بدوره بتحويلها إلى لغة الآلة ومن ثَم يفهمها الحاسب ويقوم بتنفيذها.

تصنيف لغات البرمجة :

كما ذكرنا سابقاً فى مفهوم البرمجة أن الحاسب لايفهم إلا لغة الآلة والتى تُمثل بسلسلة من الأرقام ٠ و ١ ويقوم بتنفيذها مباشرة بلا وسيط ، فبرمجة الحاسب بلغة الآلة شىء بالغ التعقيد حيث كان التعامل مع الحاسب فى بداياته فى أربعينيات القرن الماضى يحتاج إلى كتابة التعليمات بلغة الآلة وهو ما كان يتطلب وقت وجهد كبير وكعادة أى شىء يصنعه الإنسان يبدأ بسيطاً فى مهمته معقداً ومُجهِداً فى التعامل معه ثم يتطور إلى أن يصبح معقداً فى مهمته بسيطاً فى التعامل معه ، فكان من الضرورى أن يتم اختصار الوقت والجهد المبذول فى إنجاز المهام التى تحتاج إلى كتابتها بلغة الآلة إلى ما هو أهم فتم تطوير لغات البرمجة والتى استبدلت النظام الثنائى بكلمات قريبة الفهم من لغة الإنسان ، وكلما تطورت لغات البرمجة أصبحت أكثر وضوحاً فى مفرداتها وسهولة ومتعة فى كتابتها وقوة فى أدائها فأصبحت أكثر قرباً من الإنسان وبُعداً من الآلة ، بمعنى أبسط فكلما تطورت لغات البرمجة كلما ارتقت مفرداتها لذا يفهمها الإنسان ويستخدمها بسهولة وعلى العكس تصبح أكثر صعوبة لفهم الآلة لذا تحتاج إلى مترجمات بالغة التعقيد لتستطيع أن تحول المفردات القريبة من اللغة الطبيعية إلى لغة الآلة.

وتنقسم لغات البرمجة بشكل عام إلى :

- لغات منخفضة المستوى **Low level language**

هى اللغات القريبة من فهم الآلة والتى يمكن أن تتعامل مع الآلة بشكل مباشر بدون وسيط ، والمقصود بالوسيط هنا هو نظام التشغيل فلغات البرمجة الحديثة لا تخاطب الآلة بشكل مباشر بل تخاطب نظام التشغيل الذى يقوم بدور الوسيط بين الآلة وبين البرنامج المكتوب بهذه اللغة.

ولغات البرمجة القادرة على مخاطبة الآلة بشكل مباشر هى لغة الآلة **Machine Language** ولغة التجميع **Assembly** وهى لغة تتكون من مجموعة من الكلمات والرموز يتم تحويلها إلى لغة الآلة عن طريق ما يسمى بالمُجمع.

- لغات عالية المستوى **High level language**

هى لغات تتكون من مجموعة من الكلمات والرموز القريبة من اللغة الطبيعية وهذه اللغات تحتاج إلى برنامج يُطلق عليه المُترجم **Compiler** والذى يقوم بتحويل تعليمات هذه اللغة عن طريق مجموعة من المراحل إلى لغة الآلة حتى يفهمها الحاسب ويقوم بتنفيذها.

هذا هو التصنيف العام للغات البرمجة ، مع العلم أن لغات البرمجة فى كل تصنيف مرت بعدة أطوار ، ففى تصنيف اللغات منخفضة المستوى بدات بلغة الآلة التى تُكتب بأرقام النظام الثنائى وانتهت بلغة التجميع التى تُكتب بحروف وكلمات انجليزية قريبة من فهم الإنسان ، والتى كانت تُعد طفرة كبيرة فى تطور لغات البرمجة ، وفى تصنيف لغات البرمجة عالية المستوى بدأت بلغات مثل (فورتران – سى – باسكال) وغيرها ، إلى أن وصلت إلى لغات وتقنيات لإنتاج البرامج والألعاب بدون كتابة اكواد بالشكل التقليدى فما هى إلا مجموعة من النقرات والسحب والإفلات والخطوات المرتبة دون الدخول فى تفاصيل تُنتج لك فى النهاية برنامج أو لعبة تحتاج لعدة الآف من الأسطر البرمجية وأيام أو شهور لكتابتها ، تستطيع أن تنهيها فى ساعات وربما دقائق معدودة ، ومن أمثلة هذة التقنيات:

- برنامج Scratch
- برنامج Kodu
- تقنية PWCT
- تقنية Touch Develop

وجميع هذة التقنيات مجانية.

المفسر والمترجم :

البرنامج المكتوب بأى لغة برمجة هو عبارة عن نص ، مهما كان نوع اللغة المستخدمة والمهام المطلوب تنفيذها ، هذا النص يُطلق عليه اسم الشيفرة المصدرية Source Code ، يحتاج هذا الكود المصدرى إلى برنامج يقوم بتحويله إلى لغة الآلة وهى أيضا عبارة عن نص مكون من سلسلة من الصفر والواحد فقط ، هذا البرنامج بمثابة القاموس اللغوى والنحوى الذى يختبر صحة القواعد المكتوب بها الكود وصحة العبارات ثم يقوم بترجمة هذه التعليمات فى عدة مراحل إلى ملفات بلغة الآلة ثم يُنتج ملف قابل للتشغيل ، هذه المهام تختلف من حيث الكيفية من لغة برمجة إلى أخرى.

المفسر Interpreter

هو أداة تقوم بترجمة البرنامج إلى لغة الآلة سطر سطر ، وهذا يعنى أن البرنامج المكتوب بلغات برمجة تفسيرية يتم ترجمته إلى نص برمجى وسيط بلغة منخفضة المستوى (لغة وسيطة) وينتج ملف البرنامج وبداخله هذا الكود لذا لا يمكن تنفيذه على هذه الصورة مباشرة حيث يحتاج إلى برنامج آخر يقوم بترجمة الأكواد المكتوبة بداخله إلى لغة الآلة وقت التشغيل ، هذا البرنامج يسمى المفسر ، ففى لغة الجافا مثلا يُترجم النص البرمجى إلى كود وسيط يسمى Byte code وتُنتج ملف بامتداد Jar ، هذا البرنامج الناتج لا يمكن تشغيله وتنفيذه على أى جهاز مباشرة حيث يحتاج إلى ما يسمى آلة جافا الافتراضية Java VM والتى تعمل كمفسر للبرنامج الناتج حتى يتم تنفيذه ، وفى تطبيقات الدوت نت أيضا تعتمد على نفس الأسلوب بحيث يتم تحويل الكود البرمجى المكتوب بلغة من لغات الدوت نت مثل Visual Basic إلى لغة وسيطة منخفضة المستوى تسمى لغة مايكروسوفت الوسيطة IL وتُنتج ملف تنفيذى بامتداد exe ، هذا البرنامج الناتج لا يمكن تشغيله على أى جهاز مباشرة إلا إذا تم تثبيت ما يسمى إطار عمل دوت نت NET Framework. والذى يعمل كمفسر للبرامج المكتوبة بلغات الدوت نت ويقوم بترجمتها إلى لغة الآلة حسب الطلب.

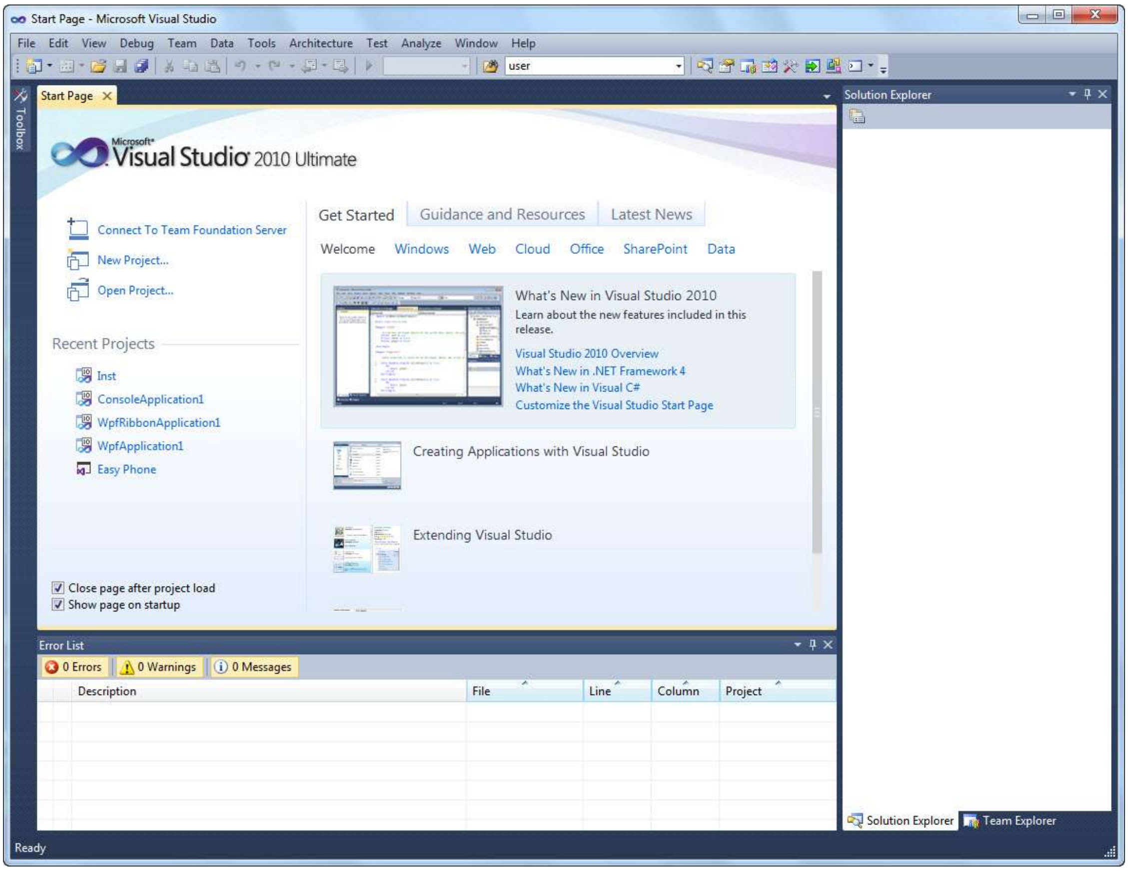
المترجم Compiler

هو أداة تقوم بترجمة البرنامج دفعة واحدة إلى لغة الآلة وتُنتج ملف تنفيذى بامتداد exe يعمل مباشرة بدون وسيط ، وهذا يعنى أن لغات البرمجة المترجمة تقوم بتحويل النص البرمجى المكتوب بها إلى لغة الآلة مباشرة وتُنتج ملف تنفيذى يعمل مباشرة دون الحاجة إلى ملفات وقت التشغيل ، ومثال على ذلك لغة C/C++ تقوم مترجماتها بإنتاج ملف تنفيذى

بامتداد exe ، هذا البرنامج يحتوى بداخله على كود بلغة الآلة يتم تنفيذه مباشرة دون الحاجة إلى مفسر ، لغة Pascal أيضا تعمل بنفس الأسلوب.

بيئة التطوير IDE

كما ذكرنا سابقاً فجميع البرامج هى فى الأصل عبارة عن ملف نصى بسيط ، فى السابق كان على المبرمج أن يكتب البرنامج بنفسه فى محرر نصوص ، وكان هذا أمراً مرهقاً ويحتاج إلى وقت طويل علاوة على كثرة الأخطاء التى يقع فيها المبرمج أثناء كتابة أكواد البرنامج ، ولاختصار الوقت والجهد ولإضافة المزيد من الإمكانيات للغات البرمجة تم تطوير أدوات تساعد المبرمج على بناء البرنامج بسهولة وبسرعة أكبر ، حيث توفر تلك الأدوات محررات أكواد ذكية تكتشف الأخطاء وتُكمل التعليمات تلقائياً وقوالب جاهزة لبعض البرامج ومكونات مكتوبة مسبقاً ومصممات للنماذج تسهل على المبرمج تصميم الواجهات ، كل هذه الأدوات موجودة فيما يسمى بأدوات التطوير IDE ، فعلى سبيل المثال مجموعة لغات الدوت نت تأتى معها بيئة تطوير تسمى Visual Studio.Net



صورة لبيئة تطوير Visual Studio إصدار ٢٠١٠

مراحل تطوير النظام :

والمقصود بالنظام هنا هو أى نظام برمجى ، فأى برنامج يتم تطويره فى عدة مراحل ، تطلق عليها أيضا دورة حياة النظام ويمكن أن نطلق عليها بشكل أبسط مراحل حل المشكلة ، هذه المراحل يمكن الإستغناء عنها فى البرامج الصغيرة جدا كأن تقوم بكتابة برنامج يقوم بحساب مجموع عددين أو حساب النسبة المئوية أو ما شابه ولكن فى البرامج الكبيرة والمعقدة لا يمكن البدء فى البرنامج مباشرة دون المرور على هذه المراحل.

- التحليل **Analysis**

فى هذه المرحلة يتم فهم البرنامج من حيث المُدخلات والمُخرجات والعمليات الحسابية والمنطقية التى سيقوم بها والمهام التى سينفذها.
تخيل أنك تعمل مبرمجاً فى شركة برمجيات وطلب عميل من الشركة عمل برنامج لإجراء حسابات هندسية وإخراج تقارير بالنتائج ، فى مرحلة التحليل يحدث الآتى:
يأتى العميل إليك ويشرح لك بالتفصيل ماذا يريد ، وبناء على ما ذكره العميل تقوم أنت أو المجموعة التى ستعمل فى البرنامج بتحليل هذه التفاصيل وتحديد المطلوب بدقة من مدخلات ومخرجات وعمليات وإجراءات بشكل مفصل ومرتب تسجل هذه التفاصيل فى ملف ، هذا ما يحدث فى مرحلة التحليل بشكل عام وبدون الدخول فى تفاصيل دقيقة.

- التصميم **Design**

فى هذه المرحلة يتم كتابة خطوات البرنامج بشكل متسلسل ومنطقى ، وتستخدم فى هذه المرحلة لغة تسمى سودو كود **Pseudo code** وتسمى أيضاً لغة الخوارزميات وهى عبارة عن كود مزيف أو وهمى غير حقيقى الهدف منه ترتيب خطوات البرنامج ، وتُستخدم أيضاً لنفس الغرض ما يسمى بالمخططات أو خرائط التدفق **Flow Chart** ولكن كل هذه الأساليب تعتبر بدائية جداً ولا تصلح إلا للبرامج متناهية الصغر ، أما فى البرامج الكبيرة فيتم استخدام طريقة أخرى وهى لغة **UML** وتستخدم لعمل نماذج من المخططات لتمثيل البرامج.

- **كتابة الكود Coding**

بعد الإنتهاء من مرحلة التحليل والتصميم أصبح الأمر فى غاية السهولة حيث يتم ترجمة خطوات البرنامج التى تم تحديدها فى خطوة التصميم إلى لغة برمجة معينة.

- **التنفيذ والاختبار Implementation & Testing**

فى هذه المرحلة يتم تنفيذ البرنامج وتشغيله وتجربته ، فلو عدنا إلى المثال الخاص ببرنامج الحسابات الهندسية يتم تجربة البرنامج بإدخال العديد من القيم لإجراء عمليات حسابية مختلفة لتحديد مدى صحة النتائج وإن كان هناك أخطاء أم لا.

- **التوثيق والصيانة Maintenance & Documentation**

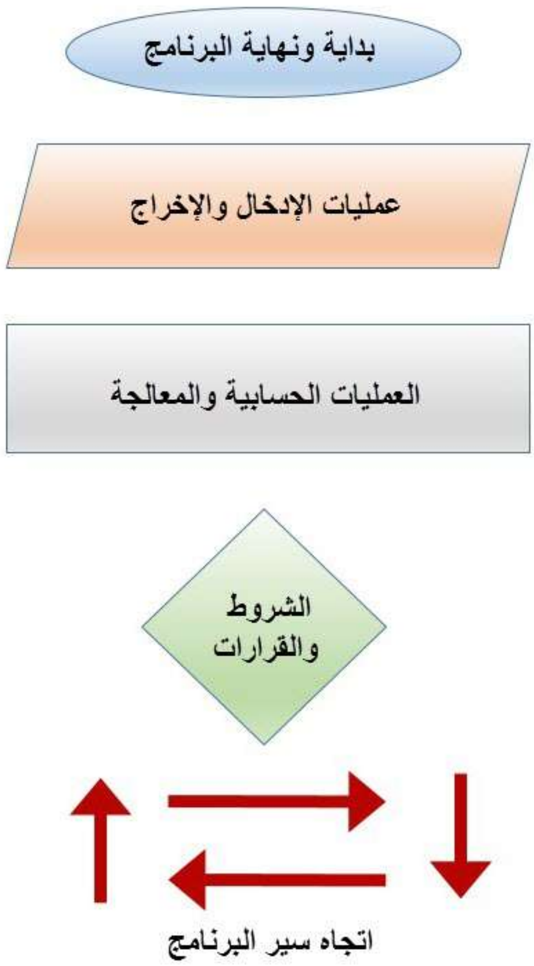
فى هذه المرحلة يتم إعداد ملفات التوثيق والتى توضح بالتفصيل كل المهام التى يقوم بها البرنامج لمساعدة المستخدم ، وأيضاً يتم تسجيل المشاكل والأخطاء التى تظهر عند استخدام البرنامج وإصلاحها وتسجيل الاقتراحات لتطوير النظام فيما بعد. فلو نظرنا إلى نظام كبير ومعقد مثل نظام التشغيل Windows وكيف تتم فيه مراحل الاختبار والصيانة ، سنجد أن شركة مايكروسوفت لا تطرح إصدارات نظام التشغيل على العملاء مباشرة بل يُطرح النظام على عدة مراحل حيث يتم طرح عدد من النسخ التجريبية للجمهور ويتم تجربة كل نسخة لمدة معينة ، فى هذه المدة تتلقى الشركة شكاوى واقتراحات العملاء وبناء عليه تقوم بإصلاح الأخطاء التى تظهر ثم تطرح إصدار تجريبى آخر إلى أن تستقر على طرح الإصدار النهائى.

فعلى سبيل المثال نظام تشغيل Windows Vista الذى صدر أواخر عام ٢٠٠٦ تم طرح عدة نسخ تجريبية منه منذ عام ٢٠٠٤ تحت اسم Longhorn إلى أن استقرت الشركة على الشكل النهائى تحت اسم Vista ولكن لم يلقى النظام قبول واسع لأسباب عديدة فقامت الشركة بإصلاح ما به من عيوب وأضافت له بعض التحسينات وأصدرته فى عام ٢٠٠٩ تحت اسم Windows 7 .

خرائط التدفق Flow Chart :

لقد ذكرنا خرائط التدفق فى مرحلة التصميم وتعرفنا على الفائدة منها وهى تمثيل لخطوات البرنامج ويمكن أن نقوم بتعريفها على أنها طريقة لتمثيل خطوات سير البرنامج باستخدام مجموعة من الأشكال الهندسية.

تستخدم فى خرائط التدفق مجموعة من الأشكال الهندسية مثل الشكل البيضاوى والمستطيل والمعين ومتوازى الأضلاع والأسهم كما فى الشكل التالى:



الأشكال الأساسية المستخدمة فى خرائط التدفق

أمثلة على خرائط التدفق:

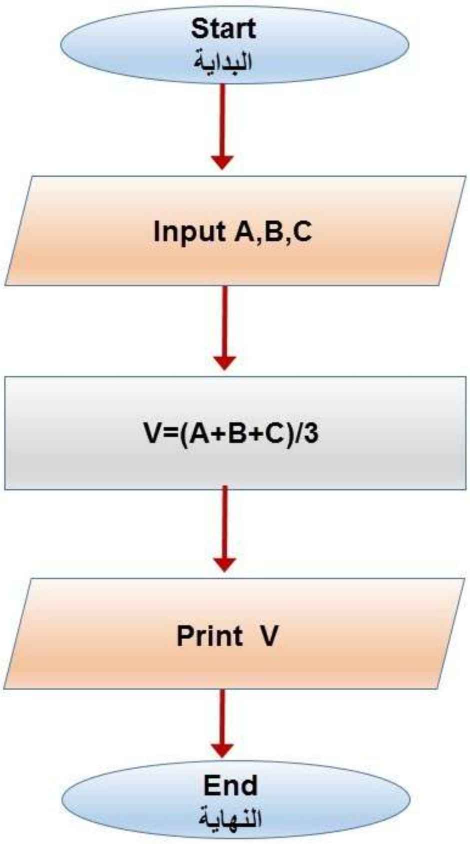
١- ارسم خريطة تدفق لبرنامج يقوم بإيجاد متوسط ثلاثة أعداد ويطبع الناتج؟

الحل:

قبل البدء فى رسم خريطة التدفق يجب عليك تحليل البرنامج أولاً

- تحديد معطيات المسألة (المدخلات) وهى ستكون عبارة عن ثلاثة أعداد سنرمز لها بالرموز A,B,C
- كتابة خوارزمية الحل أى المعادلة الرياضية التى ستخرج النتيجة المطلوبة وهى قسمة مجموع الثلاثة أعداد على عددهم وتخزين الناتج V=(A+B+C)/3
- طباعة الناتج الذى تم تخزينه فى الرمز V

وتُرسم الخريطة بهذا الشكل:



شرح الخريطة:

- تبدأ خريطة التدفق بالشكل البيضاوى الذى يرمز لبداية البرنامج
- يتم إدخال قيم المدخلات التى يُرمز لها بالرموز A,B,C وهذه الرموز تسمى فى لغات البرمجة متغيرات وهى أماكن تُحجز فى الذاكرة لتخزين أنواع معينة من البيانات

وسيتم شرحها بالتفصيل فيما بعد ويمكن أن نستخدم كلمة Input أو Enter فى عملية الإدخال داخل شكل متوازى الأضلاع

- نقوم بكتابة المعادلة الرياضية لحساب متوسط الثلاثة أعداد الذى تم إدخال قيمهم فى الخطوة السابقة وتخزين الناتج فى الذاكرة V=(A+B+C)/3 داخل شكل المستطيل
- يتم طباعة قيمة المتغير الذى تم تخزين الناتج به وهو المتغير V عن طريق الأمر Print أو Output داخل شكل متوازى الأضلاع
- يتم إنهاء البرنامج بالشكل البيضاوى بكلمة End

الآن يمكنك بكل سهولة ترجمة هذه الخطوات إلى أى لغة برمجة

وهذا هو البرنامج النهائى مكتوباً بلغة فيجوال بيزك دوت نت

```
Sub Main()
    Dim A, B, C As Integer
    Dim V As Single
    A = Console.ReadLine()
    B = Console.ReadLine()
    C = Console.ReadLine()
    V = (A + B + C) / 3
    Console.WriteLine(V)
End Sub
```

شرح أكواد البرنامج:

١- نقطة البداية : كما بدأ البرنامج فى خريطة التدفق بالشكل البيضاوى الذى يعبر عن بداية البرنامج يتم ترجمة هذة الخطوة إلى جملة Sub Main( ) والتى تعبر عن بداية تنفيذ الإجراء الرئيسى أى بداية البرنامج.

٢- تحديد وإدخال قيم المدخلات : تأتى الخطوة الثانية فى خريطة التدفق بتحديد الرموز A,B,C للتعبير عن ثلاثة مدخلات وهى الأعداد المطلوب حساب المتوسط لها ثم يتم إدخال القيم الخاصة بها عن طريق كلمة Input أو Enter ، تترجم هذه

الخطوة فى الكود إلى الأسطر من الثانى إلى السادس حيث يتم الإعلان عن حجز أماكن فى الذاكرة لثلاثة أعداد صحيحة عن طريق الجملة:

VB.NET Code

```
Dim A, B, C As Integer
```

ثم يتم حجز مكان فى الذاكرة لتخزين الناتج فى المتغير V فى السطر التالى:

VB.NET Code

```
Dim V As Single
```

بعد ذلك يتم إدخال قيم المتغيرات A,B,C فى الثلاثة أسطر التالية:

VB.NET Code

```
A = Console.ReadLine()
B = Console.ReadLine()
C = Console.ReadLine()
```

٣- **حساب المتوسط** : تأتى بعد ذلك عملية حساب المتوسط كما فى خريطة التدفق فى شكل المستطيل ويتم ترجمة هذه الخطوة إلى الكود التالى :

VB.NET Code

```
V = (A + B + C) / 3
```

٤- **إخراج الناتج** : تأتى بعد ذلك خطوة إخراج الناتج الذى تم تخزينه فى المتغير V عن طريق الجملة التالية:

VB.NET Code

```
Console.WriteLine(V)
```

٥- **إنهاء البرنامج** : تأتى بعد ذلك خطوة إنهاء البرنامج كما فى خريطة التدفق فى الشكل البيضاوى وتُترجم هذه الخطوة إلى الكود التالى :

VB.NET Code

```
End Sub
```

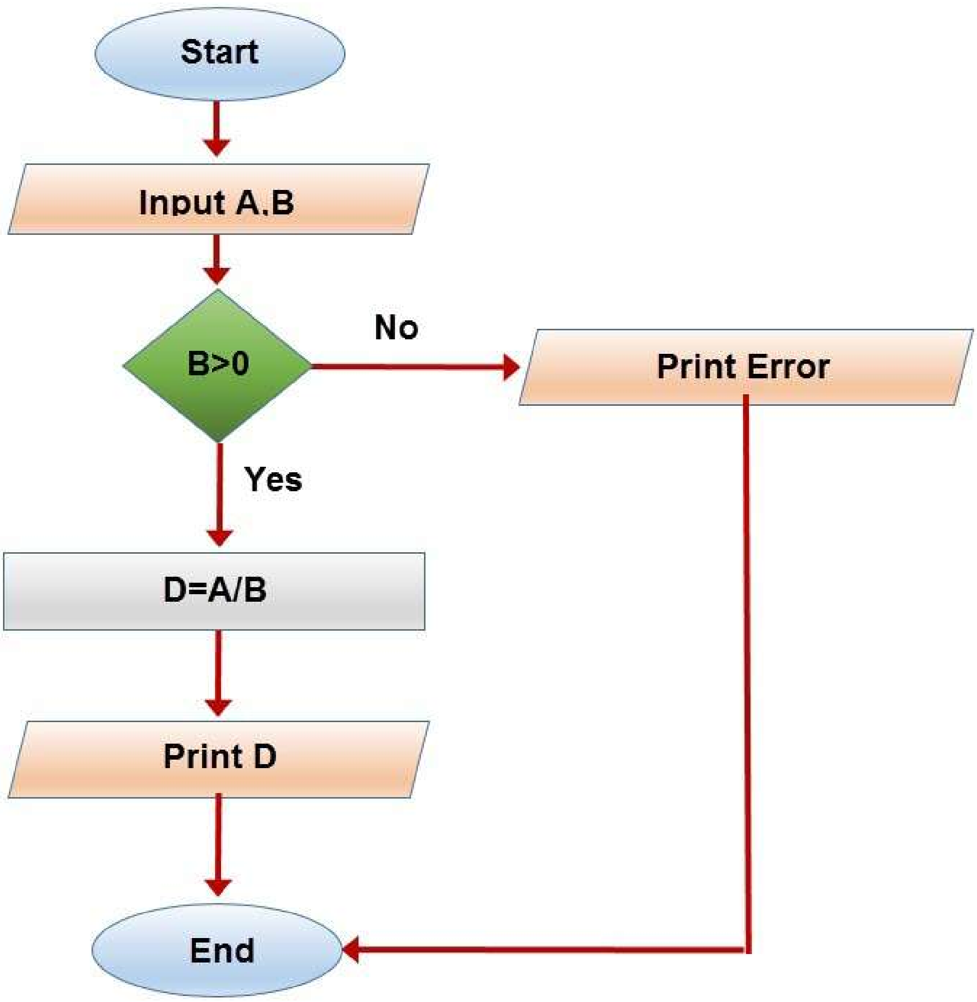
وبهذا اتضحت فائدة خرائط التدفق فهى مجرد تمثيل لخطوات البرنامج مرتبة ترتيباً منطقياً ، وعرضنا ترجمة هذا المثال إلى لغة الفيجوال بيزك لمزيد من التوضيح ولكن لا تدقق فى الأكواد كثيراً الآن ، سيتم شرحها بالتفصيل فيما بعد.

المثال الثانى:

ارسم خريطة التدفق التى توضح سير خطوات برنامج يقوم بحساب ناتج فسمة عددين؟

فى هذا المثال سنحتاج إلى إتخاذ قرار مشروط والذى يتم تمثيله فى خرائط التدفق بشكل المُعين ، حيث أنه يجب علينا اختبار قيمة العدد الثانى الذى سيأتى فى المقام والذى يجب أن يكون أكبر من الصفر لآن القسمة على صفر ليس لها معنى.

نقوم برسم خريطة التدفق بالشكل التالى:



بعد إدخال قيم المدخلات A,B يتم اختبار قيمة العدد الثانى إن كان أكبر من الصفر يتم إجراء عملية القسمة وإظهار الناتج ثم إنهاء البرنامج وإن لم يتحقق هذا الشرط فلا يجب أن تتم عملية القسمة لذا يتم تحويل مسار البرنامج إلى خطوة أخرى وهى طباعة رسالة خطأ ثم إنهاء البرنامج.

Visual Studio.Net

فيجوال ستوديو هو بيئة تطوير متكاملة أنتجتها شركة مايكروسوفت وتستخدم فى برمجة تطبيقات سطح المكتب وتطبيقات الويب والموبايل ، تدعم فيجوال ستوديو مجموعة من لغات البرمجة منها فيجوال بيزك وسى شارب وسى++ وتعمل هذه اللغات تحت منصة عمل تسمى NET Framework. والتى تحتوى على جميع الأدوات التى تستخدم فى بناء وترجمة وتشغيل البرامج.

Visual Basic.Net

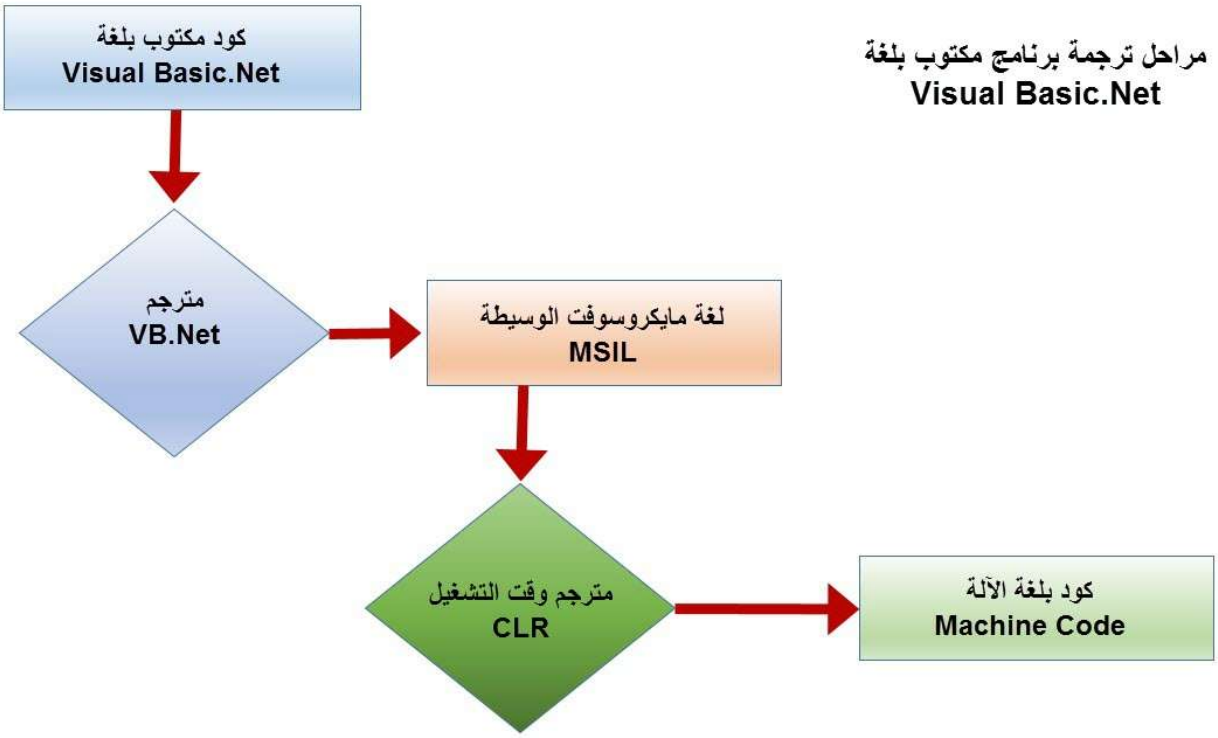
هى نسخة جديدة من لغة البرمجة الشهيرة Visual Basic ظهرت مع أول إصدار من إصدارات Visual Studio.Net فى نهاية عام ٢٠٠١ وهى تشبه إلى حد كبير لغة Visual Basic 6 وما قبلها من إصدارات حيث أن إصدار الدوت نت تم بناؤه من الصفر ولكن تم مراعاة أن تكون لها نفس القواعد وحتى نفس الكلمات المحجوزة والتعابير القديمة ظلت كما هى وأضيف إليها ، تعتبر لغة فيجوال بيسيك من اللغات السهلة فى التعلم والقوية فى الأداء ولها إنتشار كبير على مستوى العالم.

.NET Framework

هو إطار برمجى أنتجته مايكروسوفت لبناء وتشغيل البرامج ، والفكرة التى دفعت شركة مايكروسوفت لبناء هذا الإطار هو أن يكون وسيط بين البرنامج وبين نظام التشغيل وبهذا تصبح البرامج تعمل بشكل مستقل عن الآلة بحيث تعمل على أى جهاز بغض النظر عن نوع الجهاز ونظام التشغيل ، فقط يحتاج البرنامج إلى وجود إطار عمل دوت نت فريم ورك مثبتاً على الجهاز ، والفريم ورك متاح على موقع الشركة بشكل مجانى.

يتكون إطار العمل دوت نت فريم ورك من مجموعة كبيرة من المكتبات والملفات ومترجمات لغات البرمجة المدمجة معه ، وكما ذكرنا سابقاً فالبرنامج المكتوب بلغة من لغات الدوت نت يحول إلى كود بلغة وسيطة IL ويتطلب تنفيذ هذا البرنامج تثبيت نفس إصدار إطار عمل دوت نت فريم ورك الذى تم بناء البرنامج به على الجهاز المطلوب تشغيل البرنامج عليه ، وعند تشغيل البرنامج يقوم مترجم وقت التشغيل CLR بترجمة أكواد لغة مايكروسوفت الوسيطة IL إلى لغة الآلة ثم يرسلها إلى نظام التشغيل الذى يقوم بدوره بإرسالها إلى المعالج لتنفيذها.

مراحل ترجمة برنامج مكتوب بلغة
Visual Basic.Net



المحطة الثانية

أنــواع البيانــات

مقدمة

لو نظرنا إلى مراحل العمل فى الحاسب سنجد أنها عبارة عن ٣ خطوات أساسية وهى (إدخال - معالجة - إخراج) ، تُعرف المرحلة الأولى بمرحلة إدخال البيانات والمرحلة الثانية هى معالجة البيانات وإجراء العمليات الحسابية والمنطقية عليها ثم مرحلة إخراج النتائج أو المعلومات ، والبيانات هى عبارة عن قيم أوليه تُجرى عليها بعض العمليات لتصبح معلومات أى قيم لها معنى ومدلول يمكن الاستفادة منها ، إذن فجميع مراحل العمل داخل الحاسب تحتوى على بيانات والبيانات لها صور متعددة فيمكن أن تكون أرقام أو حروف أو علامات أو صور أو صوت أو فيديو ، هذه الصور جميعا يتعامل معها الحاسب على هيئة أرقام بالنظام الثنائى (لغة الآلة) وتخزن بالذاكرة أثناء فترة العمل عليها ، بناء على ما ذكرناه فإن أى برنامج يتعامل مع بيانات بحيث يستقبل بيانات من المستخدم ويرسلها للحاسب ثم يستقبل النتائج من الحاسب ويعرضها للمستخدم.

أنواع البيانات Data Types

فى لغات البرمجة إذا أردت أن تتعامل مع بيانات معينة يجب عليك أن تحجز لها مكانا فى ذاكرة الحاسب حتى يستطيع الحاسب التعامل معها ، هذه البيانات يمكن أن تكون أعداد والأعداد تختلف من حيث النوع فمنها الصحيح ومنها الحقيقى (الذى يحتوى على فاصلة عشرية) ويمكن أن يكون العدد قيمته صغيرة أو كبيرة ، والبيانات أيضا يمكن أن تكون نصوص والنص يحتوى على حروف أو أرقام أو علامات أو خليط بينهما ، من هذا الاختلاف تم تقسيم البيانات إلى أنواع فى لغات البرمجة حتى يستطيع المترجم التعرف على البيانات وإجراء العمليات المطلوبة عليها وحجز المساحة المناسبة لها فى الذاكرة حتى لا يتم إهدار المساحة المتوفرة فى الذاكرة بسرعة وتوقف عمل البرنامج أو توقف النظام بالكامل.

توجد فى فيجوال بيزك العديد من أنواع البيانات ، فالأعداد لها أنواع تخزن بها قيم صحيحة وأنواع تخزن بها قيم كسرية ، وكذلك توجد أنواع بيانات نصية ومنطقية وتاريخ ووقت وغير ذلك الكثير.

أنواع البيانات الرقمية الصحيحة فى فيجوال بيزك:

النوع	الحجم بالبايت	القيمة الصغرى	القيمة العظمى
Byte	١	٠	٢٥٥
SByte	١	١٢٨-	١٢٧
Short	٢	٣٢٧٦٨-	٣٢٧٦٧
UShort	٢	٠	٦٥٥٣٥
Integer	٤	٢١٤٧٤٨٣٦٤٨-	٢١٤٧٤٨٣٦٤٧
UInteger	٤	٠	٤٢٩٤٩٦٧٢٩٥
Long	٨	٩٢٢٣٣٧٢٠٣٦٨٥٤٧٧٥٨٠٨-	٩٢٢٣٣٧٢٠٣٦٨٥٤٧٧٥٨٠٧
ULong	٨	٠	١٨٤٤٦٧٤٤٠٧٣٧٠٩٥٥١٦١٥

أنواع البيانات الرقمية العشرية :

النوع	الحجم بالبايت	القيمة الصغرى	القيمة العظمى
Single	٤	-3.4028235E38	3.4028235E38
Double	٨	1.79769313486231E308-	1.79769313486231E308
Decimal	١٦	يستخدم لتمثيل الأعداد الصحيحة والعشرية ويمكن تحديد الدقة العشرية من صفر إلى ٢٨ خانة عشرية	

أنواع البيانات النصية :

النوع	الحجم بالبايت	القيمة
Char	٢	يخزن به حرف واحد بترميز **Unicode**
String	غير محدود	يخزن به سلسلة حرفية من صفر إلى ٢ مليار حرف بترميز **Unicode**

كما يوجد أيضا أنواع أخرى من البيانات أهمها Boolean ويخزن به قيمة منطقية True/False ونوع Date ويخزن به التاريخ والوقت

كيف تحدد النوع المناسب للبيانات التى ستستخدمها فى برنامجك؟

بناء على تحديد نوع البيان يقوم المترجم بحجز المساحة اللازمة لتخزينه فى الذاكرة ، ويجب مراعاة تحديد النوع المناسب حسب القيمة التى سيتم التعامل معها فمثلا إذا كان المطلوب هو إدخال قيمة رقمية لدرجة طالب فى امتحان مادة اللغة العربية فكيف يتم اختيار النوع المناسب لهذه القمية ، ببساطة اتبع الخطوات التالية:

حدد أدنى قيمة يمكن إدخالها فى درجة اللغة العربية (صفر)

حدد أكبر قيمة يمكن إدخالها فى درجة اللغة العربية (٤٠)

حدد ما إذا كانت القيمة المطلوب إدخالها صحيحة أم عشرية (قيمة صحيحة)

إذن فالمطلوب هو حجز مساحة فى الذاكرة لتخزين عدد صحيح يقبل إدخال الأعداد من صفر إلى ٤٠

بالنظر إلى جدول أنواع البيانات الرقمية الصحيحة تجد أن النوع الأنسب فى هذه الحالة هو Byte أو SByte

لماذا لا نختار النوع Short أو Integer مثلا أو أى نوع صحيح آخر؟

لأن هذه الأنواع تحجز مساحة أكبر فى الذاكرة والقيمة المطلوب إدخالها لن تتعدى الرقم ٤٠ فليس هناك فائدة من تحديد نوع بيان أكبر بل بالعكس سيؤدى ذلك إلى إهدار المساحة المتوفرة فى الذاكرة وإبطاء تنفيذ العمليات التى ستتم على هذا البيان بدون داعى.

المتغيرات Variables

المتغير هو مكان يُحجز فى ذاكرة الحاسب لتخزين قيمة معينة هذه القيمة يمكن لها أن تتغير أثناء تشغيل البرنامج ، وهذا المتغير له اسم ونوع وحجم وقيمة.

قواعد تسمية المتغيرات:

- أن يبدأ اسم المتغير بحرف أبجدى أو علامة _ (الشرطة التحتية)
- يمكن أن يحتوى اسم المتغير على أرقام أو حروف أو علامة _
- عدم استخدام الكلمات المحجوزة للغة Keywords
- عدم استخدام أى علامات غير علامة الشرطة التحتية _
- عدم استخدام المسافات داخل اسم المتغير
- لا يجب أن يتعدى اسم المتغير ٢٥٥ حرف
- عدم استخدام اسم لمتغير آخر فى نفس مجال الرؤية

تحديد نوع المتغير:

يتم تحديد نوع المتغير من الأنواع التى تم عرضها فى الجداول السابقة أو من الأنواع الأخرى الموجودة فى لغة فيجوال بيزك دوت نت ، ويراعى عند تحديد نوع المتغير أن يكون النوع مناسب للقيمة التى سيتم إسنادها إليه وأن يكون المدى المسموح به لهذا النوع مناسب لأقصى قيمة سيتم تخزينها فيه.

مثال:

إذا أردت كتابة برنامج لحساب النسبة المئوية لمجموع طالب فى المرحلة الثانوية ، فما هى المدخلات والمخرجات الخاصة بالبرنامج وأنواع البيانات المناسبة لكل منها

مدخلات البرنامج ستكون (درجة الطالب – المجموع الكلى)

مخرجات البرنامج (النسبة المئوية)

أنواع البيانات المناسبة:

بالنسبة للمدخلات فدرجة الطالب ستكون رقم يبدأ من الصفر وينتهى عند المجموع الكلى ويمكن أن تكون الدرجة رقما صحيحا (٣٢٠) أو ربما تحتوى على جزء عشرى (٣٥٣٥) ، وبالنسبة للمجموع الكلى فسيكون رقما صحيحا ثابتا وليكن (٤١٠) فالنوع المناسب له هو **Short** أو **UShort** ودرجة الطالب ستكون من النوع **Single** لآنها يمكن أن تحتوى على جزء عشرى.

أما بالنسبة للمخرجات فستكون رقم يبدأ من الصفر وينتهى عند ١٠٠ وأيضا يمكن أن يتضمن جزء عشرى (٩٢٢٨) فالنوع المناسب له هو **Single**.

حجم المتغير (المدى المتاح له فى الذاكرة) :

يُقاس حجم المتغير بوحدة البايت **Byte** والتى تحتوى بدورها على عدد ٨ **Bit** والـ **Bit** هى أصغر وحدة تخزين فى الحاسب حيث يُخزن بها رقم (١) أو (٠) بالنظام الثنائى (لغة الآلة)

قيمة المتغير :

هى القيمة التى تُسند إليه والتى يمكن أن تتغير أثناء تشغيل البرنامج ويجب أن تقع هذه القيمة فى المدى المسموح به لنوع المتغير فمثلا متغير من نوع **SByte** من الخطأ أن تُسند له قيمة مثل ١٩٠ لآن أقصى قيمة مسموح بها لهذا النوع هى ١٢٧

الإعلان عن المتغير : Declaring Variable

يجب الإعلان عن أى متغير قبل استخدامه وعملية الإعلان تتضمن إنشاء المتغير وتحديد اسمه ونوعه وبناء على النوع المحدد للمتغير يتم حجز المساحة اللازمة فى الذاكرة

صيغة الإعلان عن المتغير:

```
Dim [Variable Name] As [Data Type]
```

تستخدم الكلمة المحجوزة Dim للإعلان عن المتغير

يتم تحديد اسم المتغير Variable Name وفق قواعد تسمية المتغيرات

ثم تكتب الكلمة المحجوزة As

ثم يتم تحديد نوع المتغير Data Type

مثال:

```
Dim Student_Mark As Short
Dim FirstName As String
Dim Status4 As Boolean
```

ويمكن الإعلان عن أكثر من متغير من نفس النوع بهذا الشكل :

```
Dim A, B, C As Byte
```

إسناد قيمة للمتغير Assignment :

يتم إسناد القيم للمتغيرات بهذا الشكل :

```
[Variable Name] = [Value]
```

يُكتب اسم المتغير ثم علامة الإسناد = ثم القيمة

والقيمة يمكن أن تكون قيمة مباشرة مثل :

```
Student_Mark = 120
```

ويمكن أن تكون تعبير Expression مثل :

```
Student_Mark = 9 * 6
```

ويمكن أن تكون قيمة من متغير آخر مثل :

```
Student_Mark = C
```

ويمكن إسناد قيمة للمتغير أثناء الإعلان عنه :

```
Dim Student_Mark As Integer = 180
```

الثوابت Constants

الثابت هو مكان يُحجز فى ذاكرة الحاسب لتخزين قيمة معينة هذه القيمة لا يمكن أن تتغير أثناء تشغيل البرنامج ، ويتم تسمية الثوابت بنفس قواعد تسمية المتغيرات.

ويتم الإعلان عن الثوابت بهذه الصورة :

```
Const [Constant Name] As [Data Type] = [Value]
```

تستخدم الكلمة المحجوزة Const للإعلان عن الثوابت ، والإعلان عن الثابت يجب أن يشمل إسناد قيمة له والسبب هو أنه لا يمكن تغيير هذه القيمة بعد الإعلان عنه

والفرق بين الثابت والمتغير فى طريقة إسناد القيمة هى أن الثابت يجب أن تضع القيمة المجردة مباشرة أو تستخدم تعبير يُنتج قيمة رقمية أو نصية ولا يمكن أن تستخدم أسماء متغيرات أخرى أو تستدعى دوال فى تعبير لإسناد قيمة للثابت.

أمثلة :

إسناد قيمة مجردة مباشرة (صحيح)

```
Const X As Byte = 90
Const FirstName As String = "Sameh"
```

إسناد قيمة من تعبير (صحيح)

```
Const Mark As Integer = 9 * 3
```

إسناد قيمة من متغير (خطأ)

```
Const Z As Single = B
```

علما بأن B هو متغير رقمى

إسناد قيمة من تعبير يحتوى على دالة (خطأ)

```
Const Total As Short = Fix(a)
```

نطاق رؤية المتغيرات والثوابت:

والمقصود بنطاق الرؤية هو الأماكن التى يكون فيها المتغير أو الثابت مرئيا بحيث يمكن استخدامه ، فإذا تم الإعلان عن متغير داخل إجراء معين (الإجراء هو جزء من الكود له اسم محدد ويتم تنفيذه باستدعاءه أو عند وقوع حدث معين) فى هذه الحالة لايمكن استخدام هذا المتغير خارج هذا الإجراء

مثال

```
Private Sub Calc()
        Dim Number1 As Integer
End Sub
```

تم الإعلان عن متغير رقمى باسم Number1 داخل الإجراء المسمى Calc ، هذا المتغير لا يمكن استخدامه خارج نطاق الإجراء Calc

أما إذا تم الإعلان عن المتغير داخل وحدة الكود الرئيسية فسيكون مرئيا داخل جميع الإجراءات التى تتضمنها هذه الوحدة

```
Module Module1
    Dim Number1 As Integer
    Sub Main()
    End Sub
    Private Sub Calc()

    End Sub
End Module
```

فى هذا المثال تم الإعلان عن المتغير داخل وحدة الكود **Module1** والتى تضم بداخلها الإجراء **Main** والإجراء **Calc** ، فى هذه الحالة يمكن استخدام المتغير فى أى مكان داخل الوحدة وأيضا داخل الإجراءات **Main** و **Calc**

العوامل Operators

تستخدم العوامل فى إجراء العمليات المختلفة على القيم كما تستخدم فى تكوين التعبيرات **Expressions** عن طريق الربط بين المتغيرات والثوابت والتعبيرات الأخرى ، وتوجد فى لغة فيجوال بيزك دوت نت عدة أنواع من العوامل منها:

- عوامل التخصيص **Assignment Operators**
- العوامل الحسابية **Arithmetic Operators**
- عوامل المقارنة **Comparison Operators**
- عوامل التسلسل **Concatenation Operators**
- العوامل المنطقية **Logical Operators**

عوامل التخصيص:

يستخدم العامل = لتخصيص قيمة لمتغير أو خاصية معينة بالصيغة التالية:

```
Property = Value
VariableName = Value
```

الجانب الأيسر من عبارة التخصيص يحتوى على اسم الخاصية أو المتغير المطلوب تخصيص قيمة له ، والجانب الأيمن يحتوى على أى شىء له قيمة مناسبة لنوع المتغير أو الخاصية ، أى يمكن أن يحتوى الجانب الأيمن من عبارة التخصيص على قيمة مجردة أو خاصية أو ثابت أو متغير أو تعبير.

أمثلة:

```
A = 10
B = A
C = (B * A) - 2
TextBox1.Text = "Alazhar"
MyName = Label1.Text
```

العوامل الحسابية Arithmetic Operators

تستخدم العوامل الحسابية لإجراء العمليات الحسابية المختلفة

العامل	الوظيفة	الصيغة
+	الجمع	A+B
-	الطرح	A-B
*	الضرب	A*B
/	القسمة	A/B
\	قسمة الأعداد الصحيحة	A\B
Mod	باقى القسمة	A Mod B
^	الرفع لقوة (الأس)	A^2

ويتوفر أيضاً فى لغة فيجوال بيزك دوت نت عوامل تسمى بعوامل التعيين الحسابى وهى خليط بين العوامل الحسابية وعامل التخصيص وهى اختصار لعمليات رياضية شائعة

التعبير الحسابى	التعبير باستخدام عوامل التعيين الحسابى
a=a+1	a += 1
b=b-6	b -= 6
c=c*3	c *= 3
d=d/2	d /= 2
x=x\7	x \= 7

عوامل المقارنة Comparison Operators

تقوم عوامل المقارنة بمقارنة تعبيرين وإعادة قيمة منطقية إما True أو False

العامل	المعنى
=	يساوى
<>	لا يساوى
>	أكبر من
<	أصغر من
>=	أكبر من أو يساوى
<=	أصغر من أو يساوى

مثال:

التعبير a>b لو افترضنا أن قيمة a هى ١٠ وقيمة b هى ٧ فإن نتيجة عملية المقارنة ستكون True

أما التعبير x<>z فلو كانت قيمة x هى ٩ وقيمة z هى ٩ أيضاً فإن نتيجة عملية المقارنة ستكون False

عوامل التسلسل Concatenation Operators

تقوم هذه العوامل بربط عدد من السلاسل معاً لتكون سلسلة واحدة ويستخدم العامل & والعامل + لإجراء هذه العملية

لربط سلسلتين حرفيتين لتكوين سلسلة واحدة:

```
Dim MyName As String
MyName = "Alazhar" & "Alsharief"
```

أعلنا عن متغير نصى باسم MyName وأسندنا له قيمتين نصيتين تم دمجهما باستخدام العامل &

بعد عملية الإسناد ستصبح قيمة المتغير تساوى "AlazharAlsharief" ويمكن أيضاً أن نستخدم العامل + فى هذه الحالة ، ويمكن دمج قيم متغيرات نصية بالتعديل على المثال السابق سيصبح بالشكل التالى:

```
Dim MyName, S1, S2 As String
S1 = "Alazhar"
S2 = "Alsharief"
MyName = S1 + S2
```

العوامل المنطقية Logical Operators

تقوم العوامل المنطقية بمقارنة التعبيرات المنطقية وإعادة نتيجة منطقية ، ولتوضيح ذلك فبالعودة إلى عوامل المقارنة التى ذكرناها سابقاً فإن التعبير x>y هو تعبير منطقى استخدمنا فيه عامل المقارنة > للمقارنة بين متغيرين هما x و y

فإذا كانت قيمة المتغير x أكبر من قيمة المتغير y فإن نتيجة عملية المقارنة ستكون True وستكون النتيجة False إذا كانت قيمة المتغير x أقل من أو تساوى قيمة المتغير y

فإذا أردنا مقارنة تعبيرين منطقيين أو أكثر نستخدم العوامل المنطقية ومنها:

العامل	مثال	المعنى
And	`x>y And x<>20`	إذا كانت قيمة التعبير x>y هى True وقيمة التعبير x<>20 هى Trueأيضاً فإن نتيجة المقارنة بين التعبيرين ستكون True وفيما عدا ذلك ستكون النتيجة False
Or	`x<z Or a>=c`	نتيجة هذا التعبير ستصبح فى جميع الحالات True فيما عدا حالة واحدة وهى أن تكون قيمة التعبيرين هى False
Not	`Not a>b`	عامل Not هو عامل أحادى يقوم بنفى القيمة التالية له ، فإذا كانت قيمة التعبير a>b هى True ستكون النتيجة False والعكس صحيح

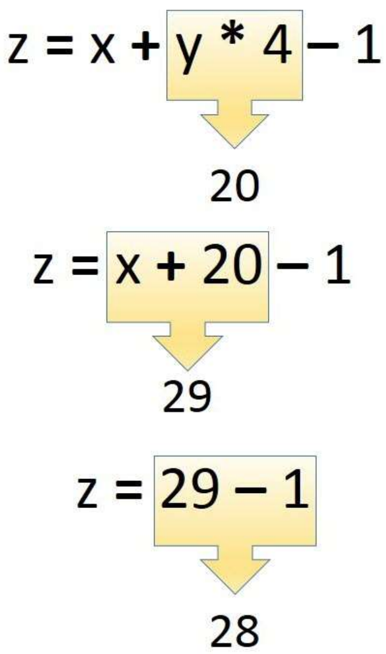
أولويات تنفيذ العمليات الحسابية:

العامل	المعنى
()	العمليات التى تقع بين أقواس وإذا تداخلت الأقواس يتم تنفيذ الأقواس الداخلية ثم الخارجية
^	عمليات الرفع للقوة (الأس)
* /	عمليات الضرب والقسمة بأولوية من اليسار إلى اليمين
\	عملية قسمة الأعداد الصحيحة
Mod	عملية إيجاد باقى القسمة
- +	عمليات الجمع والطرح بأولوية من اليسار إلى اليمين

مثال:

```
Dim x, y, z As Integer
x = 9
y = 5
z = x + y * 4 - 1
```

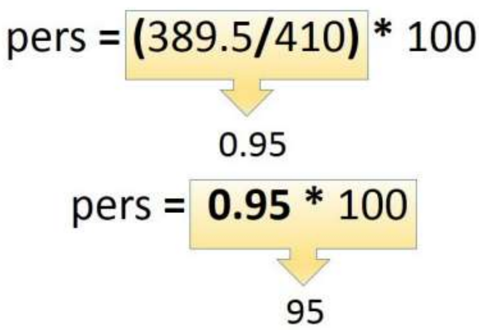
ستكون أولوية التنفيذ بهذا الشكل:



مثال لحساب النسبة المئوية لمجموع طالب:

```
Dim mark, pers As Single
Const total As UShort = 410
mark = 389.5
pers = (mark / total) * 100
```

سيتم تنفيذ العملية التى تقع بين الأقواس أولاً ثم يتم ضرب الناتج فى ١٠٠ كما فى الشكل التالى



سؤال: ما هى النتيجة فى حالة عدم استخدام أقواس لحساب النسبة المئوية بحيث تكون العملية الحسابية بهذا الشكل :

```
pers = mark / total * 100
```

؟

المحطة الثالثة

بناء الواجهة الرسومية للتطبيقات باستخدام

Windows Forms

مقدمة

قبل أن نتعرف على واجهة المستخدم الرسومية **User Interface** يجب أن نتطرق إلى مفهوم يسمى البرمجة بالكائنات **Object Oriented Programming(OOP)** والذى يُعتبر قلب لغة البرمجة **Visual Basic.Net** وهو عبارة عن نمط للبرمجة قائم على وحدات تسمى كائنات **Object** هذه الكائنات يتم بناؤها من تصنيفات أو فئات **Classes** ، والكائن هو بمثابة برنامج قابل للاستخدام أما الفئة **Class** هى عبارة عن الكود البرمجى المصدرى الذى بُنى عليه الكائن ، والكائن **Object** له خصائص **Properties** ووظائف أو وسائل **Methods** وأحداث **Events**

الخصائص هى عبارة عن متغيرات تحدد شكل وملامح الكائن وتُكتب على الصورة التالية:

```
ObjectName.Property=Value
```

حيث **ObjectName** هو اسم الكائن و **Property** هى الخاصية و **Value** هى القيمة التى يتم إسنادها للخاصية حسب نوعها ، فهناك خصائص نصية **Text** من نوع **String** وهناك خصائص رقمية من نوع **Integer** وغير ذلك من أنواع البيانات

الوسائل **Methods** هى الوظائف التى يقوم بها الكائن وتُكتب على الصورة التالية:

```
ObjectName.Method
```

أما الأحداث **Events** فهى الأفعال التى تقع على الكائن ويستجيب لها ، كأن يضغط المستخدم بزر الفأرة على الكائن فهذا الفعل يسمى حدث وبناء على هذا الحدث يقوم الكائن بتنفيذ بعض التعليمات عند وقوع هذا الحدث

ويتم تعريف الكائنات بصورة مشابهة لتعريف المتغيرات بالشكل التالى:

```
Dim ObjectName As New ClassName
```

حيث **ClassName** هى اسم الفئة التى سيُبنى منها الكائن.

نظرة سريعة على مكونات بيئة التطوير Visual Studio :

كما ذكرنا سابقاً فإن بيئة التطوير هى أداة مساعدة لتسهيل استخدام لغات البرمجة وإنتاج التطبيقات بسرعة أكبر ، وتوفر بيئة التطوير مجموعة من الأدوات مثل:

محرر الكود **Code Editor** : وهو محرر ذكى يساعد المبرمج على كتابة الكود ويقوم بأعمال مثل الإكمال التلقائى للكود وتنظيمه واكتشاف الأخطاء.

مصمم النماذج Form Designer : وهو عبارة عن أداة تسهل عملية تصميم واجهة التطبيق باستخدام مجموعة من العناصر الرسومية ، ويقوم مصمم النماذج بتوليد الكود المكافىء للتصميم بشكل تلقائى.

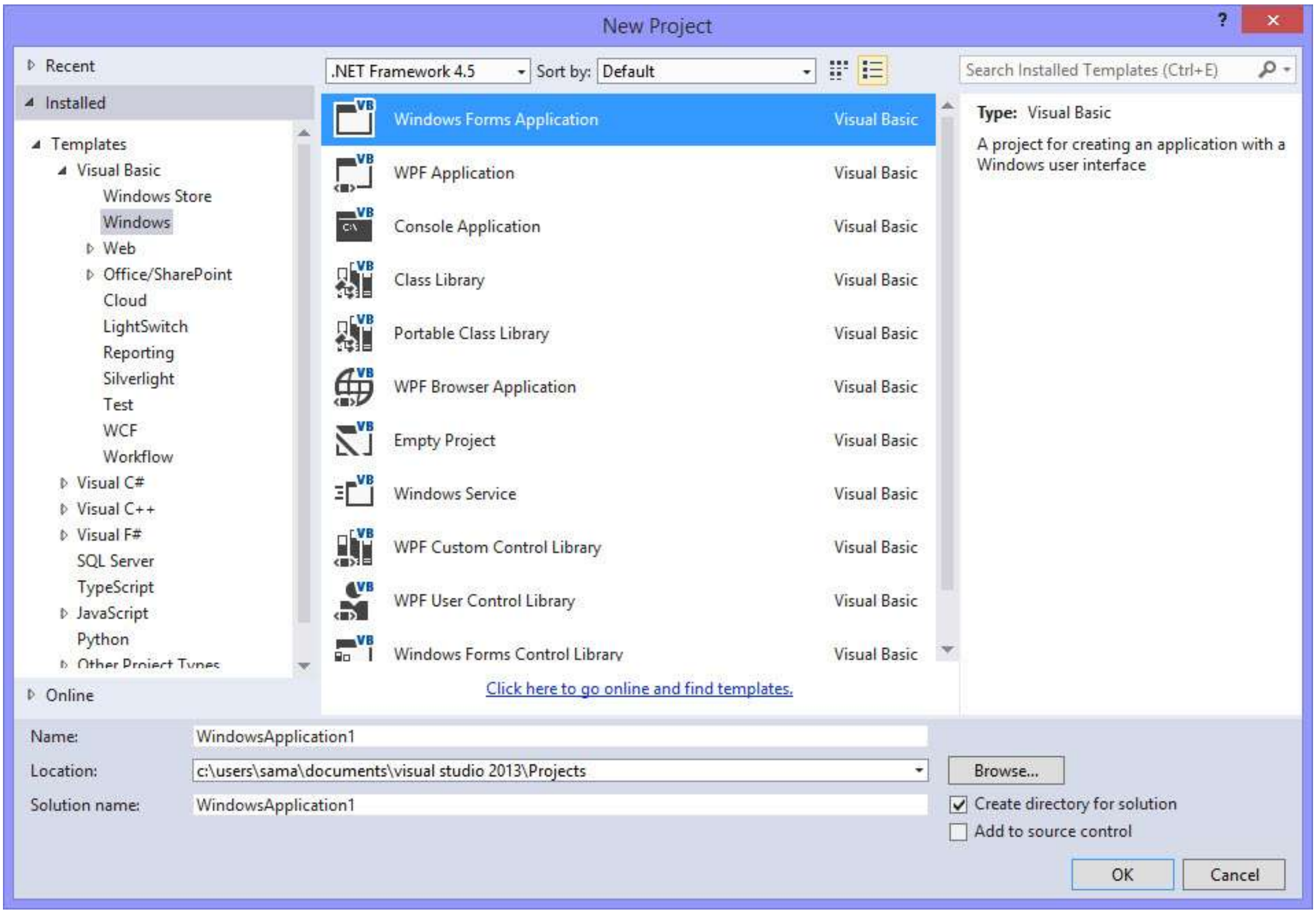
خطوات إنشاء مشروع جديد بواجهة رسومية:

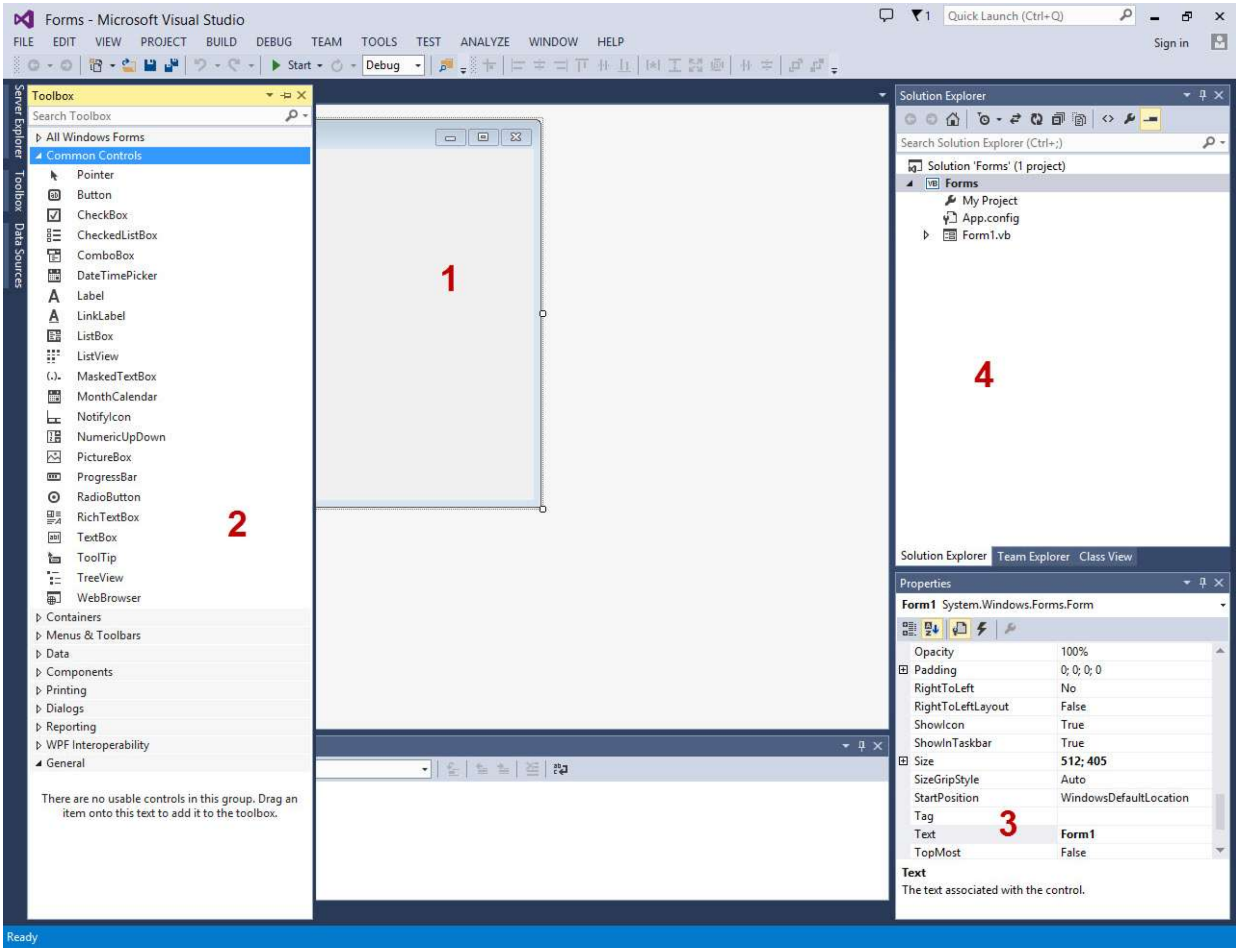
١- افتح بيئة تطوير **Visual Studio**

٢- من قائمة **File** اختر **New** ثم **Project**

٣- حدد نوع المشروع من مجموعة من القوالب **Templates** فاختر القالب **Windows Forms Application**

٤- وفى مربع **Name** أكتب اسم المشروع ثم اضغط **Ok**





مكونات بيئة التطوير عند بناء مشروع بواجهة رسومية

عند إنشاء مشروع بواجهة رسومية تظهر بيئة التطوير بهذا الشكل وتتكون من مجموعة من النوافذ كما هو موضح بالصورة:

١- نافذة النموذج Form : وهى جزء من مصمم النماذج ويتم وضع جميع الأدوات المطلوبة لبناء الواجهة الرسومية على هذه النافذة

٢- صندوق الأدوات Tool Box : وهو صندوق يعرض جميع الأدوات المتاحة لرسم واجهة المستخدم

٣- نافذة الخصائص Properties Window : وهى النافذة التى يمكن من خلالها تغيير خصائص الأدوات

٤- نافذة المشروع Solution : وهى نافذة تعرض جميع مكونات المشروع من نماذج وملفات وفئات

النموذج Form :

النموذج هو المُكون الرئيس لتطبيقات Windows ، وهو عبارة عن كائن تم بناؤه من الفئة Forms والموجودة داخل فئات NetFramework. فى المسار التالى System.Windows.Forms

والنموذج هو عبارة عن إطار توضع عليه جميع الأدوات Controls المستخدمة فى بناء واجهة المستخدم الرسومية Graphical User Interface(GUI)



شكل النموذج Form

خصائص النموذج Form Properties

تنقسم خصائص النموذج Form أو أدوات التحكم Controls إلى قسمين ، القسم الأول يمكن تغييره من خلال نافذة الخصائص Properties Window والقسم الثانى لا يمكن تغييره إلا من خلال الكود.

من أهم خصائص النموذج التى يمكن تغييرها من خلال نافذة الخصائص:

اسم الخاصية	الوظيفة
Text	تغيير نص شريط العنوان للنموذج وهى خاصية من نوع String
Name	اسم النموذج والذى يمكن التعامل مع النموذج من خلاله داخل الكود ولا يجب أن يتكرر هذا الاسم لأكثر من أداة
FormBorderStyle	تحدد شكل إطار النموذج
BackColor	تغيير لون الخلفية

BackgroundImage	وضع صورة كخلفية للنموذج
Icon	تحدد أيقونة للنموذج والتى تظهر فى شريط العنوان
MaximizeBox	تحدد إمكانية ظهور رمز التكبير والتصغير فى شريط العنوان وهو من نوع Boolean أى تأخذ القيم True أو False
MinimizeBox	تحدد إمكانية ظهور رمز الإغلاق المؤقت فى شريط عنوان النموذج
RightToLeft	تحدد اتجاه النافذة من اليمين إلى اليسار أو العكس
RightToLeftLayout	تحدد اتجاه تخطيط الأدوات الموجودة على النموذج من اليمين إلى اليسار أو العكس وهى مرتبطة بالخاصية السابقة
ShowIcon	تحدد إمكانية ظهور أيقونة النافذة فى شريط العنوان
Width	تحدد عرض نافذة النموذج بالبكسل وهى خاصية من نوع Integer أى تحدد بقيمة رقمية صحيحة
Height	تحدد إرتفاع نافذة النموذج
Left	تحدد المسافة التى تبعدها نافذة النموذج عن أقصى يسار الشاشة
Top	تحدد المسافة التى تبعدها نافذة النموذج عن أقصى نقطة فى قمة الشاشة
TopMost	تجعل النافذة فوق جميع النوافذ المفتوحة
WindowState	تحدد حجم النافذة وقت فتحها
Opacity	تحدد مدى شفافية نافذة النموذج

يمكن تغيير أى خاصية من هذه الخصائص عن طريق تحديد النموذج ثم الذهاب لنافذة الخصائص والتى تتكون من عمودين ، العمود الأيسر به اسم الخاصية والعمود الأيمن تُحدد به قيمة الخاصية ، حدد الخاصية المطلوب تغييرها ثم غير قيمتها من الخلية المقابلة فى العمود الأيمن.

مثال لتغيير خصائص النموذج عن طريق الكود:

```
Me.Text = "My First App"
Me.Width = 500
Me.Top = 200
Me.BackColor = Color.Red
Me.WindowState = FormWindowState.Maximized
Me.RightToLeft = Windows.Forms.RightToLeft.Yes
Me.TopMost = True
```

فى حالة التعامل مع النموذج داخل الـ Class الخاص به (أى داخل ملف الكود الخاص بهذا النموذج) نكتب الكلمة المحجوز Me بدلاً من اسم النموذج Form1

من وظائف/وسائل النموذج Form Methods :

اسم الوسيلة	الوظيفة
Show	إظهار نافذة النموذج فى حالة فتحه لأول مرة أو عندما يكون مخفياً
Hide	إخفاء نافذة النموذج
Close	إغلاق نافذة النموذج

وتُكتب بالصورة التالية:

```
Me.Show()
Me.Hide()
Me.Close()
```

من أحداث النموذج Form Events :

اسم الحدث	المعنى
Load	يقع هذا الحدث عند تحميل النموذج فى الذاكرة
Shown	يقع هذا الحدث عند ظهور النموذج بعد تحميله فى الذاكرة
Activated	يقع هذا الحدث عند تنشيط نافذة النموذج
Click	يقع هذا الحدث عن الضغط فى أى مكان داخل النموذج بزر الفأرة الأيسر
MouseMove	يقع هذا الحدث عند تحريك مؤشر الفأرة على النموذج
SizeChanged	يقع هذا الحدث عند تغيير حجم النموذج(العرض/الارتفاع)

وهناك أحداث كثيرة أخرى.

مثال لتعديل بعض خصائص النموذج وتنفيذ بعض الوظائف عند وقوع أحداث معينة:

اضغط DoubleClick على نافذة النموذج ، تفتح نافذة الكود فى إجراء مرتبط بالحدث Load كما فى الصورة التالية:

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As EventArgs)

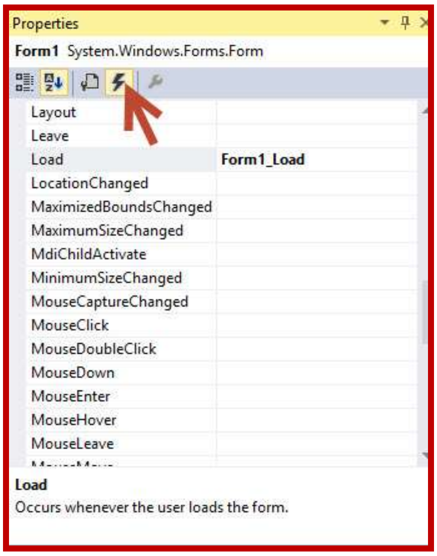
    End Sub
End Class
```

يبدأ ملف الكود الخاص بالنموذج بتعريف فئة عامة باسم النموذج Public Class Form1 وينتهى بـ End Class

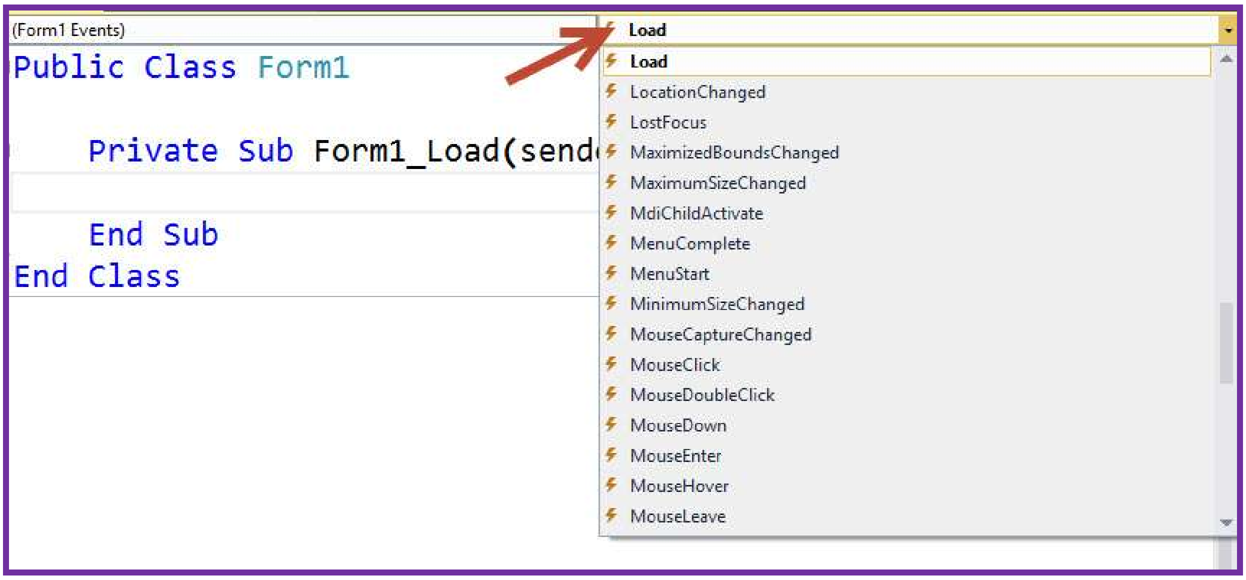
ثم يتم تعريف إجراء خاص بهذا النموذج مرتبط بحدث التحميل Private Sub Form1_Load وينتهى الإجراء بجملة End Sub ، وفى المنطقة التى تقع بين جملة Private Sub و End Sub تكتب التعليمات المطلوب تنفيذها عندما يقع حدث Load على Form1

وإذا أردت تغيير الحدث وإنشاء إجراء جديد مرتبط بحدث آخر هناك طريقتين:

الأولى من نافذة الخصائص: قم بتحديد النموذج أو أى أداة تحكم ثم اذهب إلى نافذة الخصائص واضغط على الرمز الذى الخاص بالأحداث Events ثم حدد الحدث المطلوب واضغط عليه DoubleClick يفتح نافذة الكود بإجراء جديد خاص بهذا الحدث.



الطريقة الثانية بفتح قائمة الأحداث من أعلى يمين نافذة الكود بعد تحديد النموذج أو أى أداة تحكم أخرى من القائمة الموجودة جهة اليسار



إظهار رسالة عند تغيير حجم النافذة ، قم بإنشاء إجراء جديد خاص بحدث SizeChanged لنموذج Form1 ، ثم اكتب الكود التالى:

```
Private Sub Form1_SizeChanged(sender As Object, e As
    MsgBox("Size Changed")
End Sub
```

كود تغيير لون خلفية النموذج إلى اللون الأحمر عند مرور مؤشر الفأرة عليه وتغييرها إلى اللون الأبيض عند تحريك مؤشر الفأرة خارجه

```
Private Sub Form1_MouseLeave(sender As Object, e As EventArgs) Handles Me.MouseLeave
    Me.BackColor = Color.White
End Sub

Private Sub Form1_MouseMove(sender As Object, e As MouseEventArgs) Handles Me.MouseMove
    Me.BackColor = Color.Red
End Sub
```

أدوات التحكم User Controls :

أدوات التحكم هى فى الأصل عبارة عن مجموعة من الفئات Classes داخل Windows Forms والهدف منها هو بناء واجهة المستخدم ، وتوضع أدوات التحكم على نافذة النموذج فمنها الأزرار ومربعات النصوص والقوائم وصناديق الصور وغيرها

أداة Button:

تُستخدم أداة **Button** فى جميع التطبيقات تقريباً وهى تمثل الأداة التى يضغط عليها المستخدم بالفأرة لتنفيذ أمر معين.

خصائص الـ Button :

اسم الخاصية	الوظيفة
Text	النص الذى يظهر على الزر
Enabled	تفعيل وتعطيل الزر وتأخذ القيمة **True/False**
ForeColor	لون الخط لنص الزر
Image	تُستخدم لوضع صورة داخل الزر
Visible	التحكم فى إظهار وإخفاء الزر وتأخذ القيمة **True/False**
Anchor	هى خاصية مهمة جداً وتقوم بتحديد موقع الزر عند تغيير حجم النافذة وذلك بتثبيت المسافات بين الزر وبين الحدود الخارجية للنافذة
Font	تنسيق الخط للنص الذى يظهر على الزر

مثال لتغيير خصائص الزر عن طريق الكود:

```
Button1.Visible = False
Button1.Enabled = True
Button1.Text = "Ok"
```

وسائل الزر Button Methods :

اسم الوسيلة	الوظيفة
Show	إظهار الزر
Hide	إخفاء الزر
Focus	جلب التركيز إلى الزر بحيث تجعل الزر الأداة النشطة على النافذة

أحداث الزر Button Events :

اسم الحدث	المعنى
Click	يقع هذا الحدث عند الضغط بزر الفأرة الأيسر على الزر
DoubleClick	يقع هذا الحدث عند الضغط المزدوج على الزر

MouseMove	يقع هذا الحدث عند تحريك مؤشر الفأرة فوق الزر
MouseLeave	يقع هذا الحدث عند مغادرة مؤشر الفأرة للزر
TextChanged	يقع هذا الحدث عند تغيير نص الزر
GotFocus	يقع هذا الحدث عند جلب التركيز للزر

مثال : كود إغلاق النافذة عند الضغط بزر الفأرة على **Button1**

```
Private Sub Button1_Click(sender As Object, e As EventArgs)
    Me.Close()
End Sub
```

ملاحظة :

هناك العديد من الخصائص والوسائل والأحداث المشتركة بين معظم أدوات التحكم لذا لن يتم تكرار ما تم ذكره سابقاً وسيتم التركيز على ما يميز كل أداة.

أداة Label :

تُستخدم لعرض العناوين أى نصوص ثابتة لا يمكن تغييرها أثناء تشغيل البرنامج إلا عن طريق كود برمجى.

تشترك أداة **Label** مع أداة **Button** فى كل ما تم ذكره من خصائص ووسائل وأحداث.

مثال لتغيير لون خلفية **Label1** إلى اللون الأصفر عند مرور مؤشر الفأرة عليه:

```
Private Sub Label1_DoubleClick(sender As Object, e As EventArgs)
    Label1.BackColor = Color.Yellow
End Sub
```

أداة Text Box :

تُستخدم فى عملية الإدخال (الكتابة بلوحة المفاتيح)

تشترك أداة **Text Box** مع الأدوات السابقة فى معظم الخصائص والوسائل والأحداث وتتميز ببعض الخصائص منها:

اسم الخاصية	الوظيفة
ReadOnly	تجعل مربع النص للقراءة فقط
MultiLine	تجعل مربع النص متعدد الأسطر
MaxLength	تحدد أقصى عدد من الحروف يمكن كتابته داخل مربع النص
PasswordChar	تحدد علامة كقناع للنص وتستخدم فى كلمات المرور وهى متغير من نوع Char أى تقبل إدخال حرف واحد فقط
CharacterCasing	تحدد حالة الأحرف داخل مربع النص
ScrollBars	تحدد أشرطة التمرير التى تظهر داخل النص وهى مرتبطة بخاصية MultiLine
TextAlign	تحديد محاذاة النص داخل الأداة

من وسائل الـ TextBox Methods :

اسم الوسيلة	الوظيفة
AppendText	إضافة نص إلى النص الموجود داخل الأداة
Copy	نسخ النص المحدد
Cut	قص النص المحدد
Paste	لصق النص الموجود فى Clipboard إلى مربع النص
Clear	مسح النص الموجود داخل الأداة

والحدث الافتراضى هو **TextChanged** ويقع عند تغيير النص داخل الـ **TextBox**

مثال: جعل مربع النص **TextBox1** لا يقبل إدخال المستخدم أى يصبح للقراءة فقط

```
TextBox1.ReadOnly = True
```

أداة ListBox و ComboBox :

تُستخدم هذه الأدوات فى عرض مجموعة من العناصر وتشترك مع باقى الأدوات فى معظم الخصائص والوسائل والأحداث ، ولكل عنصر فى هذه الأدوات ترتيب يُسمى **Index** وهذا الترتيب يبدأ من الرقم صفر ، فإذا كان لدينا **ListBox** بها ١٠ عناصر فترتيب العنصر الأول هو صفر وترتيب العنصر العاشر هو ٩

وأهم خصائص هذه الأدوات:

SelectedIndex : وهى تُرجع رقم العنصر المحدد وإن لم يكن هناك عنصر محدد تُرجع القيمة ١-

Items : ومن خلال هذه الخاصية يتم إضافة عناصر إلى هذه الأدوات ومثال على ذلك:

```
ListBox1.Items.Add("Visual Basic.Net")
```

ويمكن أيضاً من خلال هذه الخاصية إضافة عناصر عن طريق نافذة الخصائص.

والحدث الافتراضى لهذه الأدوات هو **SelectedIndexChanged** ويقع هذا الحدث عندما يتغير الـ **Index** للعنصر المحدد أى عندما يتم تحديد عنصر آخر غير العنصر المحدد سابقاً

أداة PictureBox :

تُستخدم هذه الأداة لعرض صورة ومن أهم خصائصها خاصية **Image** والتى تحدد الصورة التى تظهر داخل الأداة وخاصية **SizeMode** والتى تحدد طريقة عرض الصورة داخل الأداة

أداة RadioButton :

تُستخدم هذه الأداة فى عملية الاختيار لعنصر من مجموعة من العناصر فهى لا تعمل منفردة بل تكون هناك عدة أزرار منها تختار واحدا منهم

أداة CheckBox :

تُستخدم فى عمليات الاختيار وهى تعمل منفردة وليست مرتبطة بعناصر أخرى مثل **RadioButton**

أهم خصائص هاتين الأداتين هى خاصية **Checked** والتى تقوم بوضع العلامة أو إزالتها من الأداة وتقبل قيمة منطقية إما **True** لوضع العلامة أو **False** لإزالة العلامة

مثال:

```
RadioButton1.Checked = True
CheckBox1.Checked = False
```

صناديق الرسائل Message Box :

تُستخدم صناديق الرسائل فى إظهار الرسائل إلى المستخدم وتوجد فى لغة فيجوال بيزك دوت نت طريقتين لإظهار الرسائل وهما الدالة **MsgBox** و الفئة **MessageBox**

الصيغة العامة لدالة MsgBox:

```
Public Function MsgBox( _
   ByVal Prompt As Object, _
   Optional ByVal Buttons As MsgBoxStyle = MsgBoxStyle.OKOnly, _
   Optional ByVal Title As Object = Nothing _
) As MsgBoxResult
```

Prompt : هو نص الرسالة

Buttons : أزرار الرسالة

Title : هو عنوان الرسالة

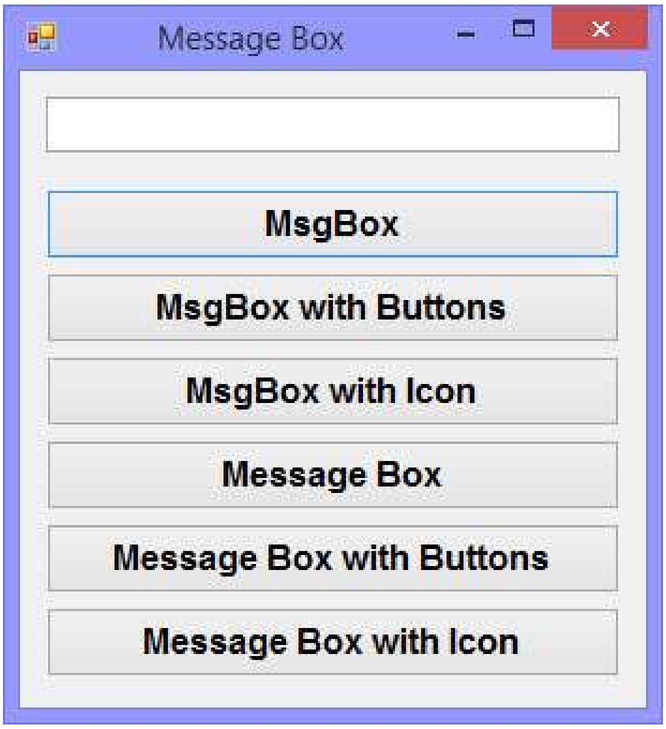
لعرض رسالة باستخدام الفئة **MessageBox** نستخدم الوسيلة **Show** بهذه الصورة:

Show(**text As String**, caption As String) As System.Windows.Forms.DialogResult
Displays a message box with specified text and caption.
text: *The text to display in the message box.*

Text : نص الرسالة

Caption : عنوان الرسالة

وهناك صور متعددة لعرض الرسائل باستخدام **MessageBox** ، ولتجربة ذلك افتح مشروعاً جديداً وصمم الواجهة بهذا الشكل:



ثم اكتب الكود التالى:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    MsgBox(TextBox1.Text)
End Sub
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    MsgBox(TextBox1.Text, MsgBoxStyle.OkCancel, "MsgBox")
End Sub
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    MsgBox(TextBox1.Text, MsgBoxStyle.YesNo + MsgBoxStyle.Information, "MsgBox")
End Sub
Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    MessageBox.Show(TextBox1.Text, "Message Box", MessageBoxButtons.YesNo, MessageBoxI
End Sub
Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    MessageBox.Show(TextBox1.Text, "Message Box", MessageBoxButtons.OKCancel)
End Sub
Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    MessageBox.Show(TextBox1.Text, "Message Box")
End Sub
```

اكتب النص الذى تريده فى مربع النص **TextBox1** ثم اضغط على الأزرار لترى رسائل بأشكال مختلفة.

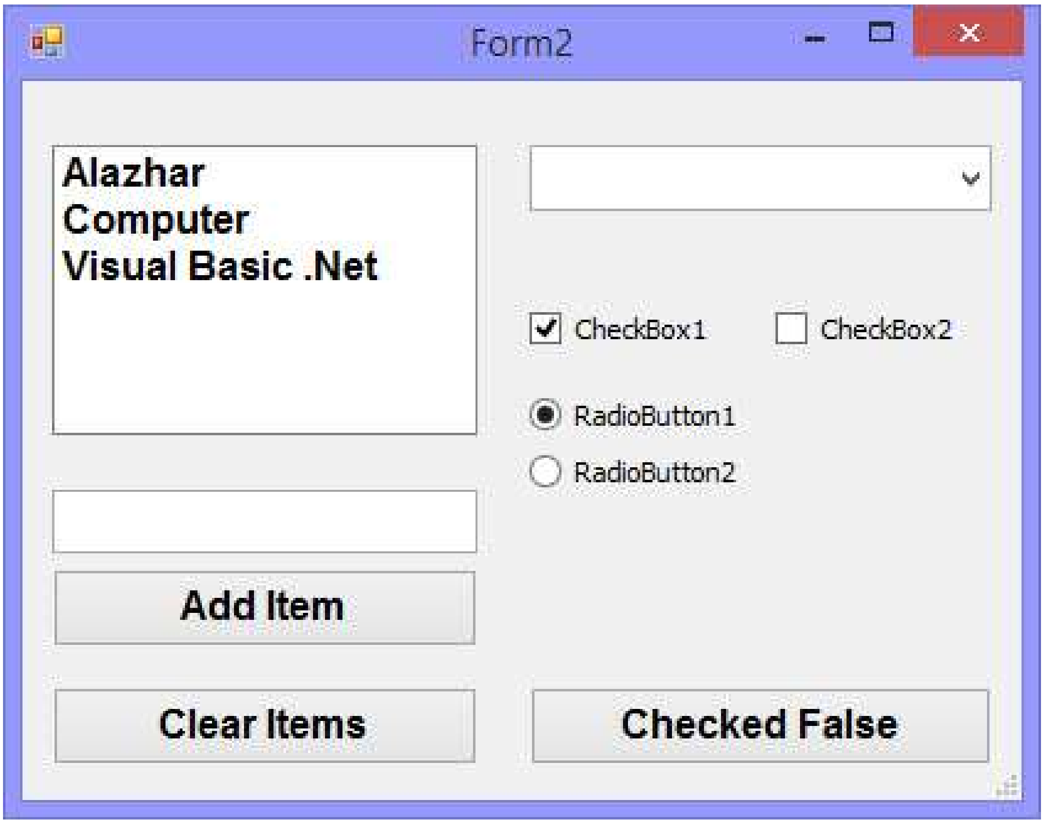
مثال على التعامل مع أدوات التحكم:

افتح مشروعاً جديداً وصمم واجهة المشروع بالشكل التالى:



ثم أضف نموذج جديد لمشروعك من قائمة **Project** اختر **Add Windows Form** ثم اختر القالب **Windows Form** ثم اضغط على **Ok**

قم بتصميم واجهة النموذج الثانى بهذا الشكل



ثم اكتب الكود التالى للنموذج الأول:

```
Private Sub Button1_Click(ByVal sender As System.Object,
    PictureBox1.Hide()
End Sub
Private Sub Button2_Click(ByVal sender As System.Object,
    PictureBox1.Show()
End Sub
Private Sub Button3_Click(ByVal sender As System.Object,
    Form2.ShowDialog()
End Sub
```

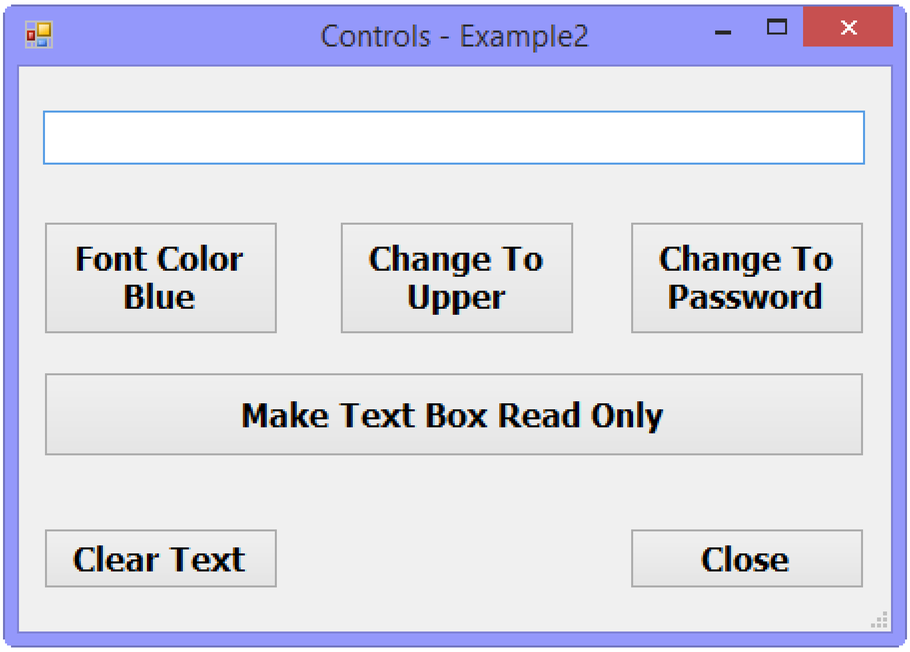
ثم اكتب الكود التالى للنموذج الثانى:

```
Private Sub Button2_Click(ByVal sender As System.Object,
    CheckBox1.Checked = False
    CheckBox2.Checked = False
    RadioButton1.Checked = False
    RadioButton2.Checked = False
End Sub
Private Sub Button1_Click(ByVal sender As System.Object,
    ListBox1.Items.Add(TextBox1.Text)
    ComboBox1.Items.Add(TextBox1.Text)
End Sub
Private Sub Button3_Click(ByVal sender As System.Object,
    ListBox1.Items.Clear()
    ComboBox1.Items.Clear()
End Sub
```

حاول أن تربط بنفسك بين الأكواد المكتوبة وبين الواجهات لمعرفة الكود المطلوب لكل أداة مرسومة على الواجهة

تمرين:

إنشىء مشروع جديد وصمم واجهته بهذا الشكل وحاول أن تكتب الكود المطلوب لتنفيذ المهام المطلوبة.



المحطة الرابعة

هياكل القرارت وحلقات التكرار

مقدمة

فى جميع البرامج تقريبا نحتاج إلى اتخاذ بعض القرارات بناء على شروط معينة وكذلك نحتاج أحياناً إلى تكرار بعض الأوامر لعدد معين من المرات بدلاً من إعادة كتابتها أكثر من مرة ، توفر لغات البرمجة ما يسمى بهياكل القرارات أو التفرع والتى تُستخدم فى تنفيذ أوامر معينة عندما يتحقق شرط معين ، وتوفر أيضاً ما يسمى بهياكل التكرار والتى تستخدم فى تكرار أوامر معينة لعدد معين من المرات.

هياكل القرارات :

تقدم لنا لغة فيجوال بيزك دوت نت مجموعة من التعليمات التى تقوم بتنفيذ تعليمة أو مجموعة من التعليمات بناء على قيمة منطقية لشرط معين ، والقيمة المنطقية هى إما **True** أو **False** والشرط هو أى تعبير له قيمة منطقية فيمكن أن يكون الشرط ناتج مقارنة بين متغيرين باستخدام عوامل المقارنة مثل **A>b** فهذا يسمى تعبير شرطى تكون نتيجته قيمة منطقية إما **True** إذا تحقق الشرط وإما **False** إذا لم يتحقق الشرط ، وتعليمات إتخاذ القرار فى فيجوال بيزك دوت نت هى :

- **If ……. Then**
- **If ……. Then …… Else**
- **Select Case**
- **Try …… Catch …… Finally**

تعليمة If …… Then

فى حالة تنفيذ سطر واحد فقط من الكود تُكتب جملة **If** على الصورة التالية :

```
If  [Conditional Exprision]  Then [Statement]
```

تُكتب الجملة على سطر واحد بحيث تبدأ بالكلمة المحجوزة **If** ثم يأتى بعدها تعبير شرطى ثم يليه الكلمة المحجوزة **Then** ثم الجملة المطلوب تنفيذها.

Conditional Exprision هو تعبير شرطى يمكن أن يكون عملية مقارنة بين قيمتين أو متغيرين أو أكثر ويمكن أن يكون متغير من نوع **Boolean** ، المهم أن تكون النتيجة فى النهاية إما **True** أو **False**

Statement هى تعليمة واحدة من الكود

أما إذا أردنا تنفيذ مجموعة من التعليمات تُكتب الجملة على الصورة التالية :

```
If  [Conditional Exprision]  Then
    [Statement 1]
    [Statement 2]
    [Statement 3]
End If
```

تُكتب التعليمات على عدة أسطر بعد كلمة **Then** ثم تُكتب الكلمة المحجوزة **End If** لإنهاء هيكل الشرط

مثال (١) :

```
Dim X, Y As Short
X = 10
If X >= 10 Then Y = X * 2
```

التعبير الشرطى هو **(X >= 10)** وهو عملية مقارنة بين متغير وقيمة مجردة باستخدام عامل المقارنة =< ، نتيجة هذا التعبير ستكون **True** لآن قيمة **X** تساوى ١٠ إذن تحقق الشرط وسيتم تنفيذ الجملة **(Y = X * 2)** المكتوبة بعد كلمة **Then** والتى تعنى إسناد القيمة الناتجة عن حاصل ضرب المتغير **X** فى العدد ٢ إلى المتغير **Y** لتصبح قيمة المتغير **Y** تساوى ٢٠

مثال (٢) :

```
Dim X, Y, Z As Integer
Z = 1
If Z <> 9 Then
     X += Z
     Y *= X
End If
```

فى هذا المثال سيتم تنفيذ تعليمتين عند تحقق الشرط **(Z <> 9)** وهما **X += Z** و **Y *= X** وستكون قيمة كل من **X,Y** تساوى ١ لان المتغير الرقمى يأخذ القيمة الافتراضية صفر إذا لم تُسند له قيمة.

تعليمة If ...... Then ...... Else

تُستخدم هذه الجملة إذا أردت تنفيذ تعليمات معينة فى حالة عدم تحقق الشرط ، وتُكتب هذه التعليمات بعد الكلمة المحجوزة **Else**

مثال على هذه الجملة : برنامج حساب النسبة المئوية لمجموع طالب

افتح مشروع جديد من Console Application ثم اكتب الكود التالى :

```
Dim Student_Mark As Single
Student_Mark = Console.ReadLine
If Student_Mark >= 50 Then
   Console.WriteLine("Student Succeeded")
Else
   Console.WriteLine("Student failed")
End If
```

تم الإعلان عن متغير لدرجة الطالب من نوع Single

السطر Student_Mark = Console.ReadLine

يعنى أن البرنامج سيقرأ القيمة التى سيدخلها المستخدم فى سطر الأوامر ويخزنها فى المتغير Student_Mark

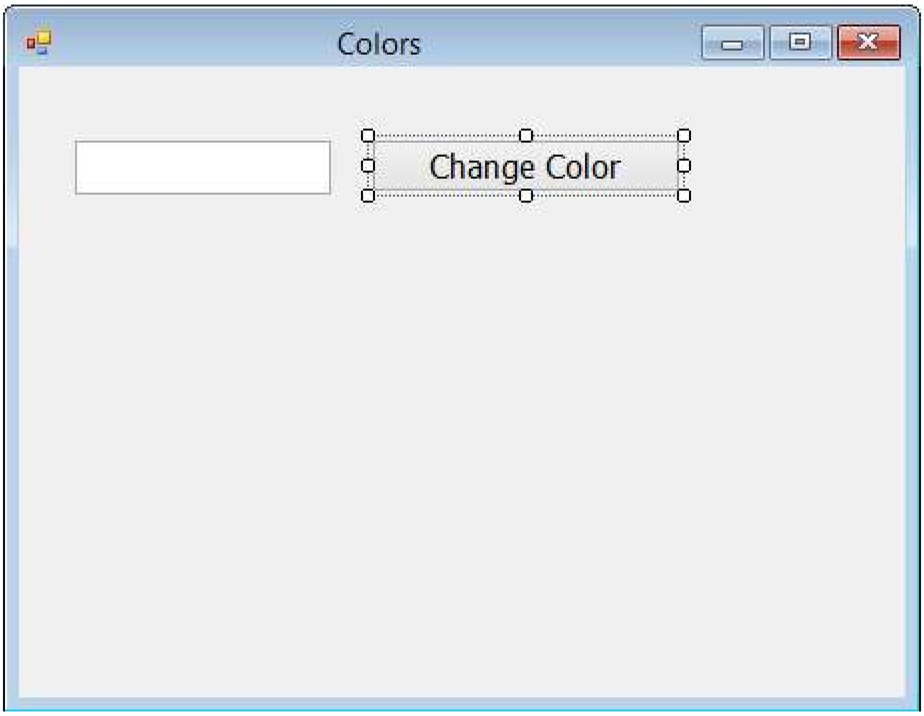
بعد ذلك تأتى جملة الشرط التى تختبر درجة الطالب إذا كانت أكبر من أو تساوى ٥٠ يطبع البرنامج الجملة Student Succeeded وهى تعنى أن الطالب نجح ، وإن لم يتحقق الشرط سيطبع البرنامج الجملة Student failed وهى تعنى أن الطالب فشل.

وفى بعض الحالات نحتاج إلى اختبار شروط متعددة بحيث أن البرنامج يختبر شرط معين فإذا تحقق هذا الشرط يقوم البرنامج بتنفيذ التعليمات المطلوبة ويخرج من الجملة الشرطية وغن لم يتحقق الشرط يقوم باختبار شرط آخر وهكذا ، نستخدم فى هذه الحالة التعليمة التالية :

```
If   [Conditional Exprision]   Then
     [Statements]
ElseIf [Conditional Exprision]   Then
     [Statements]
Else
     [Statements]
End If
```

يمكن توضيح هذه التعليمة فى هذا المثال :

افتح مشروع جديد من نوع Windows Forms Application وضع أدوات Button و TextBox على الفورم كما فى الشكل التالى



وغير خصائص الأدوات كما يلى :

الأداة	الخاصية	القيمة
Button	Name	btnColor
Button	Text	Change Color
TextBox	Name	txtColor
Form	Text	Colors

اضغط على الزر btnColor ضغطة مزدوجة بالفأرة لفتح نافذة الكود واكتب الكود التالى فى الحدث Click

```
Public Class Form1

    Private Sub btnColor_Click(sender As Object, e As EventArgs)
        If txtColor.Text = "Red" Then
            Me.BackColor = Color.Red
        ElseIf txtColor.Text = "Blue" Then
            Me.BackColor = Color.Blue
        ElseIf txtColor.Text = "Green" Then
            Me.BackColor = Color.Green
        End If
    End Sub
End Class
```

بعد تشغيل البرنامج اكتب فى مربع النص الكلمة Blue ثم اضغط على زر Change Color سيتم تغيير خلفية الـ Form لتصبح باللون الأزرق ، وإذا كتبت كلمة Red ستتغير الخلفية للون الأحمر ، وإذا كتبت كلمة Green ستتغير الخلفية للون الأخضر

ما الذى يحدث عندما تضغط على زر Change Color :

يبدا البرنامج بتنفيذ جملة الشرط ويختبر الشرط الأول "txtColor.Text = "Red فإذا كانت الكلمة المكتوبة فى مربع النص هى Red سينفذ البرنامج التعليمة التالية Me.BackColor = Color.Red والتى تقوم بتغيير خلفية النموذج إلى اللون الأحمر وينتقل البرنامج بعد ذلك إلى نهاية جملة الشرط End If ولا يختبر باقى الشروط ، اما إذا لم يتحقق الشرط ينتقل البرنامج لاختبار الشرط الثانى "txtColor.Text = "Blue فإذا كانت الكلمة المكتوبة فى مربع النص هى Blue يتحقق الشرط وينفذ البرنامج تعليمة Me.BackColor = Color.Blue والتى تغير خلفية النموذج إلى اللون الأزرق ثم يخرج البرنامج من جملة الشرط وهكذا .

ويمكن كتابة هذا البرنامج بطريقة أخرى بهذا الشكل :

```
If txtColor.Text = "Red" Then Me.BackColor = Color.Red
If txtColor.Text = "Blue" Then Me.BackColor = Color.Blue
If txtColor.Text = "Green" Then Me.BackColor = Color.Green
```

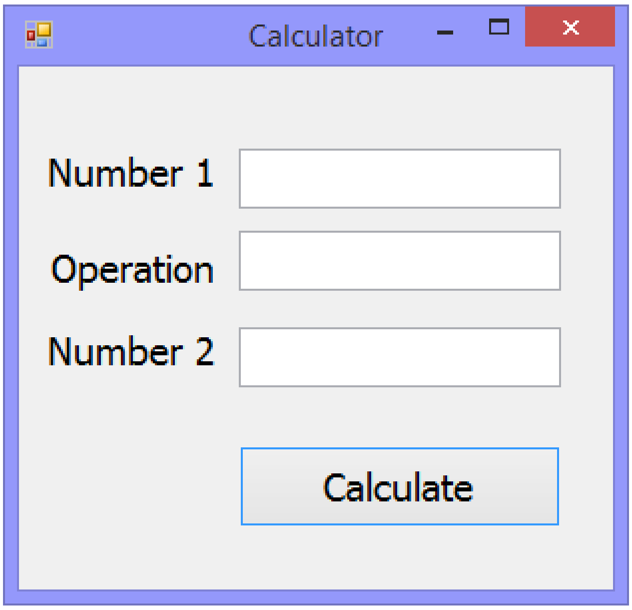
فما الفرق بين الحالتين وأيهما أفضل؟

تعليمة Select Case

تستخدم عبارة Select Case لمقارنة تعبير أو متغير واحد بقيم متعددة وتختلف عنها عبارة ElseIf فى أنها تقارن تعابير مختلفة فى كل مرة.

ويمكن توضيح وظيفة العبارة Select Case من خلال هذا المثال :

افتح مشروع جديد باسم SimpleCalculator واجعل واجهة البرنامج بهذا الشكل



غير خاصية **Name** لأدوات الـ **TextBox** لتصبح على الترتيب **txtNum1 – txtOperation – txtNum2**

غير خاصية **Name** للزر لتصبح **btnCalc** وغير خاصية **Text** لتصبح **Calculate**

ثم افتح نافذة الكود واكتب الكود التالى فى حدث **Click** للزر **btnCalc**

```
Private Sub btnClac_Click(sender As Object, e As EventArgs)
    Dim calc_Operator As String = txtOperation.Text
    Dim Num1 As Single = txtNum1.Text
    Dim Num2 As Single = txtNum2.Text
    Dim Result As Single
    Select Case calc_Operator
        Case "+"
            Result = Num1 + Num2
        Case "-"
            Result = Num1 - Num2
        Case "*"
            Result = Num1 * Num2
        Case "/"
            Result = Num1 / Num2
    End Select
    MsgBox(Result)
End Sub
```

شرح الكود :

تم الإعلان عن متغير باسم Calc_Operator من نوع String يُخزن به المعامل الحسابى الذى سيتم إدخاله فى مربع النص الثانى txtOperation

تم الإعلان عن متغير باسم Num1 من نوع Single يُخزن بداخله العدد الذى سيتم إدخاله فى مربع النص الأول txtNum1

تم الإعلان عن متغير باسم Num2 من نوع Single يُخزن بداخله العدد الذى سيتم إدخاله فى مربع النص الثالث txtNum2

تم الإعلان عن متغير باسم Result يُخزن بداخله ناتج العملية الحسابية

نبدأ بكتابة عبارة Select Case يليها المتغير calc_Operator الذى نريد اختبار قيمته فى عدة حالات

ثم نكتب بعد ذلك كل حالة بكلمة Case تليها القيمة المطلوب مقارنتها بقيمة المتغير calc_Operator ثم نكتب التعليمات المطلوب تنفيذها عندما تكون نتيجة المقارنة True أى عندما تتساوى القيمة التى تم إدخالها فى مربع النص txtOperation مع القيمة المكتوبة بعد كلمة Case

وهكذا حتى تنتهى العبارة بكلمة End Select

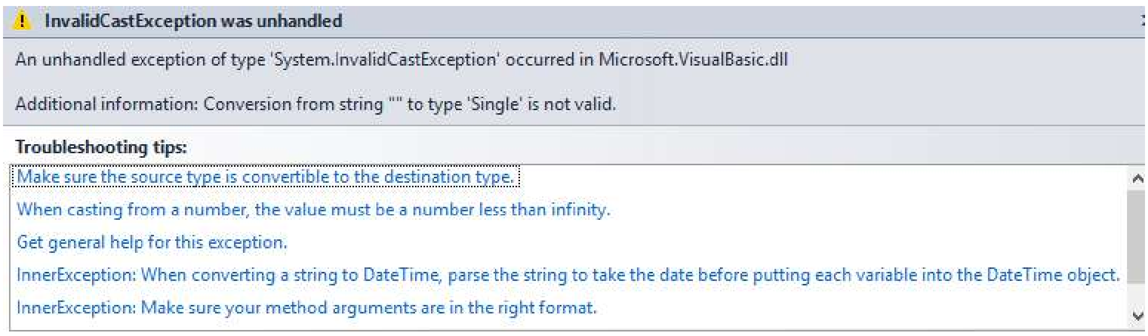
ثم نكتب كود إظهار رسالة تعرض قيمة المتغير Result

قم بتشغيل البرنامج وادخل قيم عددية فى مربعات النص الأول والثالث وأدخل المعامل الحسابى فى مربع النص الثانى ثم اضغط على زر Calculate لإظهار النتيجة فى رسالة

ملاحظة:

فى حالة عدم إدخال المعامل الحسابى فى مربع النص الثانى ستظهر الرسالة وبها القيمة صفر وذلك لآن عبارة Select Case لن ينفذ منها شىء لآنها لن تتساوى قيمة مربع المعامل الحسابى مع أى من حالات العبارة Select Case

أما فى حالة عدم إدخال قيم فى أى من مربعات النص الخاصة بالأعداد ستظهر رسالة الخطأ التالية :



نص رسالة الخطأ

```
Conversion from string "" to type 'Single' is not valid
```

وهذا يعنى أن المترجم غير قادر على تحويل القيمة الفارغة الموجودة فى مربع النص الأول إلى النوع Single ليتم تخزينها فى المتغير Num1

ولتفادى هذا الخطأ يمكننا استخدام عبارة Try ...... Catch

تستخدم العبارة Try Catch لمعالجة الاستثناءات ، بمعنى أنه من الممكن حدوث أخطاء استثنائية سواء كانت هذه الأخطاء متوقعة ومعلومة أو غير معلومة ، كما فى المثال السابق إذا أدخل المستخدم القيم المطلوبة بشكل صحيح سيتم تنفيذ البرنامج بدون أخطاء وإذا حدث استثناء ولم يُدخل المستخدم إحدى القيم فى هذه الحالة سيحدث خطأ ، فتستخدم العبارة Try Catch لاصطياد تلك الأخطاء ، وتُكتب الكلمة Try تليها التعليمات المطلوب تنفيذها ثم تُكتب كلمة Catch تليها التعليمات المطلوب تنفيذها فى حالة حدوث أخطاء ، ثم تليها كلمة End Try لإنهاء العبارة

ويمكن إظهار رسالة للمستخدم بالخطأ وذلك بالإعلان عن متغير بعد الكلمة Catch بالشكل التالى

ex As Exception

حيث ex هو اسم المتغير و Exception هو نوعه

ويمكن إظهار نص رسالة الخطأ للمستخدم من خلال الخاصية Message للمتغير ex

فيصبح الكود بالشكل التالى :

```
Private Sub btnClac_Click(sender As Object, e As EventArgs)
    Try
        Dim calc_Operator As String = txtOperation.Text
        Dim Num1 As Single = txtNum1.Text
        Dim Num2 As Single = txtNum2.Text
        Dim Result As Single
        Select Case calc_Operator
            Case "+"
                Result = Num1 + Num2
            Case "-"
                Result = Num1 - Num2
            Case "*"
                Result = Num1 * Num2
            Case "/"
                Result = Num1 / Num2
        End Select
        MsgBox(Result)
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try

End Sub
```

الحلقات التكرارية :

تُستخدم حلقات التكرار لتنفيذ مجموعة من التعليمات لعدد معين من المرات بناء على شرط محدد ، وتقدم لغة فيجوال بيزك دوت نت العديد من حلقات التكرار.

حلقة Do … Loop :

تُستخدم حلقة Do … Loop لتكرار تنفيذ مجموعة من التعليمات لعدد غير محدد من المرات بناء على القيمة المنطقية للشرط المستخدم ، ويستمر تنفيذ الحلقة التكرارية طالما تحقق الشرط أي عندما تكون قيمته True ، والشرط كما ذكرنا سابقا هو أي تعبير تكون له قيمة منطقية.

الصورة العامة لحلقة Do … Loop :

```
Do While Condition
    Code Block
Loop
```

مثال :

طباعة الأعداد من ١ إلى ١٠ على الشاشة

افتح مشروع جديد من نوع Console Application وقم بتغيير اسمه إلى Numbers1To10 ، ثم اكتب الكود التالى :

```
Sub Main()
    Dim N As Byte = 1
    Do While N <= 10
        Console.WriteLine(N)
        N += 1
    Loop
    Console.ReadLine()
End Sub
```

شرح الكود :

فى البداية تم الإعلان عن متغير من نوع Byte ليكون عداد للحلقة وتم إعطاءه القيمة الابتدائية ١

```
Dim N As Byte = 1
```

بدأت الحلقة بالكلمات المحجوزة Do تليها كلمة While يليها الشرط المطلوب تكرار تنفيذ الحلقة إذا كانت نتيجته True ، والشرط هنا عبارة عن عملية مقارنة متغير بقيمة محددة باستخدام معامل المقارنة =< ، ومعنى هذا أن الحلقة لن تبدأ فى التنفيذ إذا كانت نتيجة هذا الشرط False

نكتب الجملة التالية لطباعة قيمة المتغير N على الشاشة السوداء

```
Console.WriteLine(N)
```

والتى ستطبع القيمة الابتدائية للمتغير N وهى ١

ثم نقوم بزيادة قيمة المتغير N بمقدار ١ عن طريق كتابة التعبير التالى

```
N += 1
```

لاحظ أننا استخدمنا معامل التعيين الحسابى =+ بدلا من كتابته على هذه الصورة N=N+1

الآن أصبحت قيمة المتغير N تساوى ٢
ثم نكتب الكلمة المحجوزة Loop والتى تعيد التنفيذ لبداية الحلقة فيتم اختبار الشرط مرة أخرى وبناء على القيمة الناتجة من الشرط يتم التنفيذ أو يتم الخروج من الحلقة إذا كانت القيمة الناتجة هى False

اما السطر التالى

```
Console.ReadLine()
```

فالفائدة منه هو إبقاء الشاشة السوداء مفتوحة لحين الضغط على مفتاح Enter
بعد تشغيل البرنامج ستجد أنه طبع الأعداد من ١ إلى ١٠ على الشاشة

استخدام Until بدلاً من While فى حلقة Do .. Loop

عند استخدام كلمة While مع حلقة Do .. Loop يتم اختبار الشرط قبل الدخول إلى الحلقة بينما يمكن اختبار الشرط بعد تنفيذ الحلقة لمرة واحدة على الأقل باستخدام الكلمة المحجوزة Until وتُكتب الحلقة على هذه الصورة:

Do
Code Block
Loop Until Condition

نبدأ بكتابة كلمة Do ثم التعليمات المطلوب تكرار تنفيذها ، ثم نقوم بتكرار التنفيذ باستخدام كلمة Loop بناء على قيمة التعبير الشرطى المكتوب بعد كلمة Until ، الاختلاف الثانى بين الحلقتين هنا حيث يتم تكرار تنفيذ الحلقة إذا كانت القيمة الناتجة عن الشرط هى False ويتم إنهاء الحلقة إذا كانت قيمة الشرط True

المثال السابق باستخدام كلمة Until :

```
Sub Main()
    Dim N As Byte = 1
    Do
        Console.WriteLine(N)
        N += 1
    Loop Until N > 10
    Console.ReadLine()
End Sub
```

فى هذه الحلقة سيتم تنفيذ التعليمات مرة واحدة على الأقل قبل اختبار قيمة الشرط تغير التعبير الشرطى فبدلاً من N<=10 عند استخدام الكلمة **While** فى بداية الحلقة ، أصبح N>10 عند استخدام الكلمة **Until** فى نهاية الحلقة

ملاحظات هامة :

- يمكن استخدام كلمة **While** فى نهاية الحلقة لاختبار الشرط بعد تنفيذ التعليمات مرة واحدة على الأقل ، ويمكن أيضاً استخدام الكلمة **Until** فى بداية الحلقة لاختبار الشرط قبل الدخول فى الحلقة ، جرب كتابة الحلقة باستخدام كلمة **While** فى نهاية الحلقة ومرة أخرى باستخدام كلمة **Until** فى بداية الحلقة وحاول أن تستنتج الفرق بينهما.
- يمكنك استخدام عبار **Exit Do** للخروج من الحلقة فى أى وقت إذا تحقق شرط معين.
- يمكن أن تستمر الحلقة التكرارية إلى ما لانهاية إذا لم يتحقق الشرط المطلوب لإنهاء الحلقة ، ففى المثال السابق مثلاً إذا لم تقم بزيادة قيمة المتغير N ستدخل الحلقة فى عملية تكرارية لا نهائية وستطبع الرقم ١ فى كل مرة.

الحلقة التكرارية For ... Next :

تُستخدم الحلقة التكرارية **For ... Next** لتكرار تنفيذ مجموعة من التعليمات لعدد محدد من المرات معلوم مسبقاً على عكس حلقة **Do** والتى لا يُعرف فيها عدد مرات التكرار مُسبقاً ، وفى حالة معرفة عدد مرات التكرار المطلوبة من المُفضل استخدام حلقة **For Next** بدلاً من حلقات **Do** والصورة العامة لحلقة **For .. Next** بهذا الشكل:

```
For Counter=Start  To  End  [Step]
    Code Block
Next Counter
```

تستخدم حلقة **For** متغير عددى كعداد للحلقة **Counter** تحدد له قيمه ابتدائية **Start** وقيمة نهائية **End** ، ثم تحدد قيمة الخطوة أى مقدار الزيادة **Step** وإذا لم يتم تحديدها تكون قيمة الزيادة الافتراضية هى ١ ، ثم يتم تكرار الحلقة بكلمة **Next** يليها اسم متغير العداد **Counter**

مثال:

طباعة الأعداد الفردية من ١ إلى ١٠٠

افتح مشروع جديد من نوع **Console Application** وقم بتغيير اسمه إلى **PrintFrom1To100** ثم اكتب الكود التالى:

```
Sub Main()
    Dim N As Byte
    For N = 1 To 100 Step 2
        Console.WriteLine(N)
    Next N
    Console.ReadLine()
End Sub
```

بدانا بالإعلان عن متغير عداد الحلقة وسميناه **N** من نوع **Byte**

ملاحظة: يمكن تسميه متغير العداد بأى اسم مناسب ، أما اختيار النوع **Byte** فهو النوع الأنسب فى هذه الحالة حيث أننا لن نحتاج إلا إلى تخزين القيم العددية من ١ إلى ١٠٠ فى هذا المتغير فليس هناك حاجة لاستخدام متغير أكبر فى المدى مثل **Short** أو **Integer**

بدأت الحلقة بالكلمة المحجوزة **For** ثم تحديد القيمة الابتدائية للمتغير بـ ١ والقيمة النهائية بـ ١٠٠ ، ثم تم تحديد قيمة الخطوة بـ ٢ وذلك حتى يتم طباعة الأعداد الفردية فقط ، قم نكتب جملة طباعة قيمة المتغير على الشاشة ثم كلمة **Next** والتى تقوم بزيادة قيمة متغير العداد بمقدار القيمة المحددة فى **Step** ثم تعيد تنفيذ الحلقة حتى تصل قيمة العداد إلى قيمة أكبر من القيمة النهائية وهى ١٠٠

ويمكن أيضاً الخروج من الحلقة فى أى وقت باستخدام الكلمة **Exit For**

مثال على حلقات التكرار:

برنامج لطباعة حاصل ضرب عددين حسب الأعداد التى يتم إدخالها باستخدام حلقات التكرار

افتح مشروع جديد من نوع **Windows Forms Application** وقم بتصميم الواجهة بهذا الشكل:



قم بتغيير خاصية **Text** للفورم إلى **New Calculator**

أضف أداة **ListBox** وزر **Button** وقم بتغيير خاصية **Text** له إلى **Start** وغير خاصية **Name** إلى **btnStart** ثم قم بفتح نافذة الكود فى حدث **Click** للزر **btnStart** واكتب الكود التالى:

```
Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles
    Dim Num1, Num2, Result As Single
    Try
        Do
            Num1 = InputBox("Input First Number", "Input Number")
            Num2 = InputBox("Input Second Number", "Input Number")
            ListBox1.Items.Add(Num1 & "*" & Num2 & "=" & Num1 * Num2)
        Loop Until Num1 = Nothing Or Num2 = Nothing
    Catch ex As Exception
        MsgBox("Program Canceld")
    End Try
End Sub
```

شرح الكود:

```
Dim Num1, Num2, Result As Single
```

- الإعلان عن ثلاثة متغيرات من نوع Single لآننا نريد إيجاد حاصل ضرب أى عدد سواء كان صحيحاً أو به جزء عشرى لذا كان الاختيار الأنسب للنوع Single
- استخدمنا عبارة Try لتفادى الخطأ وستظهر الفائدة من استخدامها عند تنفيذ البرنامج
- استخدمنا حلقة Do وكلمة Until فى نهاية الحلقة وذلك لكى يتم تنفيذ الحلقة مرة واحدة على الأقل حتى يمكننا اختبار الشرط الذى يفترض أن صندوق الإدخال InputBox قد ظهر بالفعل وتم الضغط على زر Cancel لإرجاع قيمة "لا شىء" Nothing

```
Num1 = InputBox("Input First Number", "Input Number")
```

- نقوم بتعيين قيمة للمتغير Num1 عن طريق استدعاء صندوق الإدخال InputBox لإدخال قيمة عددية للرقم الأول
- ثم نستدعى صندوق الإدخال مرة أخرى لتعيين قيمة للعدد الثانى وتخزينها فى المتغير Num2

```
Num2 = InputBox("Input Second Number", "Input Number")
```

- فى السطر التالى نقوم بطباعة عملية الضرب فى أداة ListBox ، واستخدمنا العامل التسلسلى & لضم مجموعة من القيم مع بعضها البعض لتظهر ككتلة واحدة

```
ListBox1.Items.Add(Num1 & "*" & Num2 & "=" & Num1 * Num2)
```

- ثم تأتى مرحلة اختبار الشرط باستخدام كلمة Until

```
Loop Until Num1 = Nothing Or Num2 = Nothing
```

والذى يختبر القيمة التى سيتم إرجاعها من صندوق الإدخال فى حالة الرقم الأول أو الرقم الثانى إن كانت Nothing تنتهى الحلقة ، ولاحظ أننا استخدمنا فى هذا الشرط العامل المنطقى Or والذى يقارن بين تعبيرين منطقيين ويُرجع قيمة منطقية وكما ذكرنا سابقاً ففى جميع الحالات سيُرجع قيمة True إلا فى حالة واحدة فقط وهى أن تكون قيمة التعبيرين هى False ، والمطلوب فى هذا الشرط للخروج من الحلقة هو أن تكون النتيجة True أى عندما تكون قيمة أى من التعبيرين True أو كليهما ومعنى

أن تكون قيمة التعبير True هى أن يضغط المستخدم على زر Cancel فى صندوق الإدخال فيُرجع القيمة Nothing وبالتالى تصبح نتيجة التعبير True

وفى حالة الضغط على زر Cancel لإنهاء الحلقة يحدث الخطأ التالى

Conversion from string "" to type 'Single' is not valid.

ويحدث هذا الخطأ لأن المتغير المطلوب إسناد القيمة له من نوع Single والقيمة المطلوب إسنادها إليه من صندوق الإدخال هى من نوع String ففى حالة إن كانت هذه القيمة Nothing يحدث هذا الخطأ فتتلقاه Catch وتظهر رسالة بإنهاء البرنامج وتتوقف عملية الضرب

ويمكن تفادى هذا الخطأ بطرق أخرى منها على سبيل المثال تحويل القيمة الموجودة فى صندوق الإدخال إلى قيمة رقمية قبل إسنادها إلى المتغير باستخدام الدالة Val كما يلى

```
Num2 = Val(InputBox("Input Second Number", "Input Number"))
```

فى هذه الحالة لن يحدث خطأ عندما نضغط على زر Cancel فى صندوق الإدخال

المحطة الخامسة

الدوال والإجراءات

الإجراءات Procedures :

الإجراءات هى عبارة عن مجموعة من التعليمات داخل إطار من الكود يبدأ بعبارة إعلان وينتهى بعبارة End ، ويتم استدعاء الإجراء من مكان آخر داخل الكود ، عندها يتم تنفيذ التعليمات الموجودة فى الإجراء ثم ينتقل التحكم إلى السطر الذى تم فيه استدعاء هذا الإجراء.

ويوجد فى لغة فيجوال بيزك دوت نت عدة أنواع من الإجراءات منها:

- إجراءات Sub والذى يقوم بتنفيذ مجموعة من التعليمات ولا يُعيد قيمة إلى كود الاستدعاء.
- إجراءات Function والتى تقوم بتنفيذ مجموعة من التعليمات ثم تُعيد قيمة إلى كود الاستدعاء.
- إجراءات معالجة الأحداث Event Handling والتى تقوم بتنفيذ مجموعة من التعليمات استجابة لحدث معين.

الفائدة من استخدام الإجراءات:

تُفيد الإجراءات فى حالة المهام التى نحتاج إلى تنفيذها مرات عديدة فى البرنامج ، وتساعد الإجراءات فى تقسيم البرنامج وتنظيم الكود مما يجعله أكثر وضوحا فى القراءة والمراجعة لاكتشاف الأخطاء بسهولة ، كما يمكن استخدم إجراءات تم كتابتها فى برامج أخرى.

الإجراء Sub :

هو عبارة عن مجموعة من التعليمات التى تبدأ بالكلمة المحجوزة Sub وتنتهى بـ End Sub ، وعند استدعاء الإجراء يتم تنفيذ التعليمات الموجودة به ولا يُعيد أى قيمة إلى كود الاستدعاء ، ويتم كتابة الإجراء بالصورة التالية:

```
Sub (Procedure Name)(Parameters)
   Code Block
End Sub
```

يتم كتابة اسم الإجراء بعد كلمة Sub ثم يتم تحديد ما يُسمى Parameters بين قوسين وهى اختيارية.

ما هى الـ Parameters ؟

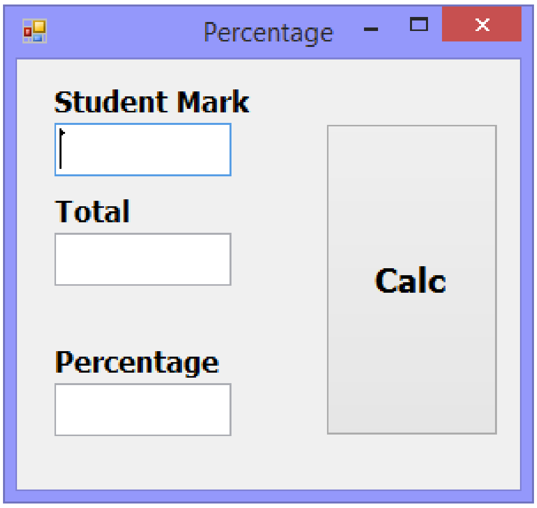
هى متغيرات يتم تمريرها أثناء استدعاء الإجراء لتستخدم فى تنفيذ التعليمات الموجودة داخله

جملة Code Block تعنى مجموعة التعليمات المطلوب تنفيذها عند استدعاء الإجراء ثم ينتهى الإجراء بكلمة End Sub ويعود التحكم لسطر الاستدعاء

مثال:

برنامج حساب النسبة المئوية لمجموع طالب

افتح مشروع جديد من نوع Windows Forms Application وقم بتصميم الواجهة بهذا الشكل:



ثم قم بإنشاء إجراء باسم CalcPercentage واكتب الكود التالى:

```
Sub CalcPercentage(ByVal Mark As Single, ByVal Total As Short)
    txtResult.Text = ((Mark / Total) * 100).ToString
End Sub
```

فى هذا الكود تم إنشاء الإجراء وتم تحديد Parameters لتمريرها للإجراء أثناء استدعاءه وهما Mark من نوع Single والذى ستخزن فيه درجة الطالب ، والثانى هو Total من نوع Short والذى سيخزن فيه المجموع الكلى ، ثم تم كتابة العملية الحسابية التى تقوم بحساب النسبة المئوية مستندة إلى الـ Parameters التى تم تحديدها للإجراء وهى قيم مجهولة حتى الآن ، ينتظر الإجراء حتى يتم استدعاءه وعندها سيتم تمرير قيم لهذه الـ Parameters والتى يحتاجها الإجراء لحساب النسبة المئوية

فى حدث Click للزر btnCalc اكتب الكود التالى:

```
Private Sub btnCalc_Click(sender As Object, e As EventArgs)
    Dim Mark As Single = Val(txtMark.Text)
    Dim Total As Short = Val(txtTotal.Text)
    CalcPercentage(Mark, Total)
End Sub
```

فى السطر الأول تم الإعلان عن متغير من نوع **Single** وتم إسناد القيمة الموجودة فى مربع النص **txtMark** إليه

وفى السطر الثانى تم الإعلان عن متغير من نوع **Short** وتم إسناد القيمة الموجودة فى مربع النص **txtTotal** إليه

هذه المتغيرات هى التى سيتم تمرير قيمها إلى الإجراء أثناء استدعاءه ليقوم بإجراء العملية الحسابية بناء عليها

ثم يتم استدعاء الإجراء بكتابة اسمه ثم فتح قوسين وكتابة الـ **Parameters** المطلوبة

تتضح أهمية استخدام الإجراءات بشكل أكبر فى التعليمات التى يتم تكرار تنفيذها كثيراً ومثال على ذلك إفرض أنك تقوم بكتابة برنامج لتسجيل بيانات معينة ولتكن بيانات الطلاب مثل (اسم الطالب – الرقم القومى – تاريخ الميلاد – الفصل – الصف - ... الخ)

هذه البيانات سيتم إدخالها فى فورم به مجموعة من أدوات الـ **TextBox** ، وسيكون هناك أزرار مثل (جديد – حفظ – خروج)

فى برنامج مثل هذا كم مرة ستحتاج إلى مسح البيانات الموجودة فى مربعات النص؟

ستحتاج إلى مسح البيانات مرتين ، مرة عند الضغط على زر جديد لمسح البيانات المعروضة لتسجيل بيانات طالب ، ومرة أخرى بعد عملية الحفظ ليتم مسح البيانات استعداداً لإدخال بيانات طالب آخر ، قم بتجربة هذا بنفسك لتعرف أهمية استخدام الإجراء فى تقليل الأكواد المكتوبة وتنظيم البرنامج.

الدوال Functions :

الدوال هى إجراءات تحتوى على مجموعة من التعليمات تبدأ بكلمة **Function** وتنتهى بكلمة **End Function** وتُكتب بالصورة التالية:

```
Function  (Function Name)(Parameters) As Data Type
        Code Block
        Return Value
End Function
```

فى الدوال تحتاج إلى تحديد نوع القيمة التى سيتم إعادتها لكود الاستدعاء

تعديل المثال السابق باستخدام الدوال **Function** بدلاً من الإجراء **Sub**

```
Function CalcPercentage(ByVal Mark As Single, ByVal Total As Short) As Single
    Return (Mark / Total) * 100
End Function

Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
    Dim Mark As Single = Val(txtMark.Text)
    Dim Total As Short = Val(txtTotal.Text)
    txtResult.Text = CalcPercentage(Mark, Total)
End Sub
```

تم تحديد نوع القيمة المُعادة من الدالة بالنوع **Single** فى سطر الإعلان ، ثم تم كتابة الكلمة المحجوزة **Return** والتى تقوم بإعادة القيمة التى تليها إلى كود الاستدعاء

يتم استدعاء الدالة بكتابة اسمها يليها المعاملات الخاصة بها **Parameters** على يمين عبارة التخصيص ومعنى ذلك أن الدالة ستصبح قيمة بعد استدعائها وهذه القيمة يجب أن يتم إسنادها إلى متغير أو ثابت من نفس النوع ، كما حدث فى المثال بعد التعديل حيث تم استدعاء الدالة على يمين عبارة تخصيص قيمة لمربع النص **txtResult** ويقوم فيجوال بيزك دوت نت بعملية تحويل تلقائى لنوع البيان الناتج عن الدالة من **Single** إلى **String** حتى يتم تخزينه فى خاصية **Text** الخاصة مربع النص **txtResult**

ما معنى كلمة **ByVal** التى تم كتابتها قبل كل **Parameter** ؟

هذه الكلمة تعنى أن الإجراء أو الدالة سيقوم باستخدام قيمة المتغيرات الممررة له ولن يؤثر على قيمها الأصلية إذا تغيرت داخل تعليمات الدالة أو الإجراء.

مثال آخر لاستخدام الدوال Function :

اكتب دالة تقوم بوظيفة المعامل الحسابى ^ والذى يقوم بوظيفة الرفع لقوة (الأس)

افتح مشروع جديد من نوع **Windows Forms Application** وقم بتغيير اسمه إلى **PowerFunction** ثم اكتب الكود التالى:

```
Function Pow(ByVal Number As Integer, ByVal Power As Byte) As Integer
    Dim N As Byte
    Dim R As Integer = 1
    For N = 1 To Power
        R *= Number
    Next
    Return R
End Function
```

تم عمل دالة باسم **Pow** ولها **Parameters** التالية

Number : وهو الرقم المطلوب رفعه لقوة وتم تحديد نوع **Integer**

Power : وهو الرقم المطلوب الرفع إليه أى الأس ونوعه **Byte**

وتم تحديد القيمة العائدة من الدالة بالنوع **Integer**

فى البداية تم الإعلان عن متغير العداد للحلقة التكرارية **For** وهو المتغير **N** من نوع **Byte**
فى السطر التالى تم الإعلان عن متغير **R** من نوع **Integer** وأعطيناه القيمة الافتراضية ١
استخدمنا حلقة التكرار **For** لتكرار تنفيذ العملية الحسابية بعدد مرات تساوى قيمة المتغير **Power**
ثم تم ضرب قيمة المتغير **R** فى قيمة المتغير **Number** وتخزين الناتج فى المتغير **R**
وبعد انتهاء الحلقة يتم إعادة القيمة المخزنة فى المتغير **R** باستخدام كلمة **Return**

يتم استدعاء الدالة عن طريق الضغط على زر **btnPower** كما فى الكود التالى:

```
Private Sub btnPower_Click(sender As Object, e As EventArgs)
    Dim Number As Integer = Val(txtNumber.Text)
    Dim power As Byte = Val(txtPower.Text)
    txtresult.text = Pow(Number, power)
End Sub
```

يتم تعريف متغيرين ليُخزن بهما القيم المُدخلة فى مربعات النص والخاصة بالرقم وقيمة الرفع للقوة ، ثم يتم استدعاء الدالة وإسناد القيمة الناتجة منها إلى مربع النص **txtResult**

بعض الدوال الهامة فى لغة فيجوال بيزك دوت نت :

الدالة Val :

تُستخدم الدالة Val فى الحصول على القيمة الرقمية من سلسلة حرفية وتُكتب بالصورة التالية:

Val(S)

حيث S هى السلسلة الحرفية والتى يمكن أن تكون نص مجرد أو متغير من نوع **String** أو خاصية **Text** لأداة معينة

مثال:

```
Val(TextBox1.Text)
```

القيمة الناتجة من الدالة Val فى عدة حالات:

القيمة الموجودة داخل TextBox1	القيمة الناتجة من دالة Val
Alazhar	0
N45	0
25	25
12mm_56	12

إذا كانت القيمة الموجودة داخل السلسلة الحرفية المطلوب تحويلها إلى رقم بدالة **Val** هى رقم يتم تحويلها إلى نفس القيمة من نوع **Double** ، أما إذا كانت نص لا يبدأ برقم تكون القيمة الناتجة صفر ، أما إذا كانت نص يبدأ برقم تكون النتيجة هى أقصى رقم من ناحية اليسار ، وفى حالة إن كانت قيمة فارغة أى **Nothing** تكون القيمة الناتجة هى صفر.

الدالة Str و ToString :

تقوم الدوال **Str** و **ToString** بتحويل قيمة رقمية إلى سلسلة حرفية ، بمعنى أن يقوم المترجم بالتعامل معها على أنها سلسلة حرفية وليست رقم يمكن إجراء عمليات حسابية عليه ، وليس معنى ذلك أنها يقوم بتحويل الأرقام إلى حروف.

الصيغة العامة لهذه الدوال:

```
Str(Number)
Number.ToString
```

والمقصود بـ Number هو إما قيمة رقمية مجردة أو متغير رقمى أو تعبير رياضى يُنتج قيمة رقمية مع ملاحظة أنه فى حالة ToString لا يمكن تحويل قيمة رقمية مجردة.

مثال:

```
Dim A As Integer = 45
Dim N, M As String
N = Str(A)
M = (A * 4 + 99).ToString
```

بعد تنفيذ البرنامج ستصبح قيمة N تساوى 45 ويراها المترجم سلسلة حرفية من نوع String ، أما M ستساوى 279 وهى ناتج إجراء العملية الحسابية A*4+99 وأيضاً يراها المترجم سلسلة حرفية من نوع String

الدالة Fix :

تُستخدم فى الحصول على الجزء الصحيح من عدد حقيقى وتُكتب بالصورة التالية

```
Fix(R)
```

حيث R يمكن أن تكون قيمة رقمية مجردة أو متغير رقمى أو تعبير رياضى ، مثل

```
Fix(A)
Fix(45.235)
Fix(C/N-1)
```

فى المثال التالى

```
Dim R As Single = 99.545
Dim N As Integer = Fix(R)
```

قيمة N ستصبح 99

الدالة Rnd :

تقوم بتوليد عدد حقيقى عشوائى أكبر من أو يساوى الصفر وأقل من الواحد الصحيح

مثال:

```
Dim R As Single = Rnd()
MsgBox(R)
```

يقوم هذا الكود بإظهار رسالة تحتوى على رقم عشوائى أكبر من أو يساوى الصفر وأقل من الواحد

الدالة Randomize :

تُكتب هذه الدالة قبل دالة Rnd وتفيد فى عدم تكرار الرقم العشوائى الناتج عن الدالة Rnd

مثال:

```
Randomize()
Dim R As Single = Rnd()
MsgBox(R)
```

وإذا أردت الحصول على رقم عشوائى صحيح استخدام دالة Fix مع ضرب القيمة الناتجة عن دالة Rnd فى أقصى رقم تريد توليده

مثال

```
Dim R As Single = Fix(Rnd() * 9)
```

هذا الكود سيولد رقم عشوائى من 0 إلى 8

كيف حدث ذلك؟

أقل قيمة ستولدها Rnd هى مثلاً 0.000001 قم بضربها فى الرقم 9 سيكون الناتج 0.000009 باستخدام دالة Fix تتحول هذه القيمة إلى 0 وهى أدنى قيمة يمكن توليدها

أقصى قيمة ستولدها Rnd هى مثلاً 0.99999 قم بضربها فى الرقم 9 ستكون النتيجة 8.99991 باستخدام الدالة Fix تتحول هذه القيمة إلى 8 وهى أقصى قيمة يمكن توليدها

الدالة IsNumeric :

تقوم هذه الدالة باختبار قيمة معينة أو تعبير معين لمعرفة ما إذا كان رقماً أم لا وتُرجع هذه الدالة قيمة منطقية إما True أو False

وتُكتب بهذه الصورة:

```
IsNumeric(Value)
```

حيث Value هى أى قيمة مجردة أو متغير أو تعبير

الدالة IsDate :

تقوم هذه الدالة باختبار قيمة معينة إن كانت تاريخ أم لا وتُرجع قيمة منطقية ، إن كانت القيمة تاريخ تُرجع True وإن لم تكن تاريخ أو تاريخ خاطىء تُرجع False

مثال:

```
Dim D As String = "1/12/2010"
If IsDate(D) Then
    MsgBox("True")
Else
    MsgBox("False")
End If
```

فى هذا المثال ستُرجع الدالة القيمة True لآن قيمة المتغير D هى تاريخ صحيح

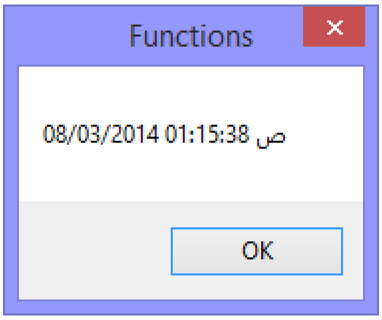
خصائص التاريخ والوقت Date And Time :

Now :

تُرجع هذه الخاصية التاريخ والوقت الحالى للنظام وتُكتب بهذا الشكل Now()

ويمكن أن تُسند قيمتها إلى أى متغير من نوع Date

```
Dim D As Date = Now
MsgBox(D)
```



الدالة Today :

تُرجع هذه الخاصية التاريخ الحالى للنظام

ويمكن أن تُسند قيمتها إلى أى متغير من نوع Date

```
Dim D As Date = Today
MsgBox(D)
```

الدالة TimeOfDay :

تُرجع هذه الخاصية الوقت الحالى للنظام

ويمكن أن تُسند قيمتها إلى أى متغير من نوع Date

```
Dim D As Date = TimeOfDay
MsgBox(D)
```

أداة Timer :

تستخدم أداة Timer لتنفيذ أو تكرار تنفيذ مجموعة من التعليمات عند مرور فترة زمنية محددة

من أهم خصائص اداة Timer

خاصية Interval : والتى تحدد الفترة الزمنية التى يأخذها الإجراء المصاحب لحدث Tick لتكرار تنفيذ التعليمات الموجودة بداخل هذا الإجراء ، وتُحدد القيمة بالمللى ثانية أى أن 1000 تعنى ثانية واحدة

خاصية Enabled : وهى المسئولة عن تفعيل وإيقاف تفعيل الـ Timer ولها قيمة منطقية إما True أو False

مثال: افتح مشروع جديد وأضف أداة Label إلى الفورم ثم أضف أداة Timer وغير خصائصها كما يلى:

```
Interval=1000
Enabled=True
```

وفى حدث Tick اكتب الكود التالى:

```
Label1.Text = TimeOfDay
```

سيقوم هذا الكود بطباعة الوقت الحالى فى أداة Label1 وسيتغير الوقت كل ثانية.

الفهرس

- اداة CheckBox
- صناديق الرسائل Message Box

المحطة الرابعة : هياكل القرارات وحلقات التكرار

- مقدمة
- تعليمة If ...... Then
- تعليمة If ...... Then ...... Else
- تعليمة Select Case
- عبارة Try ...... Catch
- حلقة Do ... Loop
- الحلقة التكرارية For ... Next

المحطة الخامسة : الدوال والإجراءات

- الإجراءات Procedures
- الدوال Functions
- الدالة Val
- الدالة Str و ToString
- الدالة Fix
- الدالة Rnd
- الدالة Randomize
- الدالة IsNumeric
- الدالة IsDate
- خصائص التاريخ والوقت Date And Time
- أداة Timer

إعداد/ سامح كامل
Email: mrsameh1@gmail.com
Facebook: www.facebook.com/mr.samehkamel